COMPUTER APPENDIX A

SYSTEM AND METHOD FOR PREDICTION OF BEHAVIOR IN COMPLEX SYSTEMS

Inventors: Amos Yahil
Richard C. Puetter

Attorney Docket No.: 7684-PA01

264 pages

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/29 14:34:03 ayahil>
SUBROUTINE Algebron( &
  & abstol, &          ! Absolute error limit in minimization
  & add_iter, &        ! Flag for adding iterations
  & chi2, &            ! Chi^2 at each sampling point
  & gof, &             ! Goodness-of-fit measure
  & gof_ev, &          ! Expectation value of the GOF measure
  & gof_std, &         ! Standard deviation of the GOF measure
  & n_t, &             ! # of samples
  & nsig, &            ! # of sigma for Lambda truncation
  & p_t, &             ! # of valid securities
  & prn, &             ! Print flag
  & prob, &            ! Probabilities of chi^2 array
  & r_t, &             ! Returns
  & reltol, &          ! Relative error limit in minimization
  & signif, &          ! Significance level for accepting an iteration
  & stepmx, &          ! Maximum # of steps in FRPRMN
  & v, &               ! Resultant covariance matrix
  & w_t, &             ! Weights
  & z &               ! Update parameters
  & )

 USE Interfaces, ONLY: Covar, Cull, Func, Get_gof, Next, Sample
 USE Mynr, ONLY: Frprmn, Linmin
 USE Nr, ONLY: Gammq
 USE Nrtype
 USE Nrutil, ONLY: Assert_eq, Get_diag, Ifirstloc
 USE Parm, ONLY: diagv, idx, iflag, k, n, norm, p, r, sm, w
 USE Utils, ONLY: Permute, Wher
 IMPLICIT NONE
                    ! Arguments
 INTEGER, INTENT(in) :: add_iter, prn, stepmx
 INTEGER, INTENT(in), TARGET :: n_t, p_t
 REAL(sp), INTENT(in) :: abstol, nsig, reltol, signif
 REAL(sp), INTENT(in), TARGET, DIMENSION(n_t) :: w_t
 REAL(sp), INTENT(in), TARGET, DIMENSION(n_t,p_t) :: r_t
 REAL(sp), INTENT(inout), DIMENSION(p_t*(p_t+1)) :: z
 REAL(sp), INTENT(out) :: gof, gof_ev, gof_std
 REAL(sp), INTENT(out), DIMENSION(n_t) :: chi2, prob
 REAL(sp), INTENT(out), DIMENSION(p_t,p_t) :: v
                    ! Locals
```

```
INTEGER :: chk, i, it, kmn, kmx, ndx, m, np, nz, pp1
INTEGER, ALLOCATABLE, DIMENSION(:) :: tdx
INTEGER, DIMENSION(p_t*(p_t+1)) :: iz
REAL(sp) :: f, f_old, nd, np_old
REAL(sp), ALLOCATABLE, DIMENSION(:) :: xi
REAL(sp), DIMENSION(p_t*(p_t+1)) :: z_old
REAL(sp), DIMENSION(p_t) :: ln, std
                    ! Initialization & pointers
n => n_t
p => p_t
r => r_t
w => w_t
IF( ALLOCATED( sm ) ) DEALLOCATE( sm )
ALLOCATE( diagv(p), sm(n) )
                    ! Check sizes
chk = Assert_eq( (/ n, SIZE(chi2), SIZE(prob), SIZE(r,1), SIZE(w) /), &
   & ' Algebron-n' )
chk = Assert_eq( p, SIZE(r,2), SIZE(v,1), SIZE(v,2) , ' Algebron-p' )
pp1 = Assert_eq( p*(p+1), SIZE(z), ' Algebron-z' )
                    ! Decrypt input z & determine # of factors
iz = Permute( pp1, seed=123571113 )
IF( ALL( z == 0.0_sp ) ) THEN
  k = 0
ELSE
  WHERE( z /= 0.0_sp )
    z = SIGN( EXP( ABS( z ) ) - iz, z )
  END WHERE
  k = (pp1 - Ifirstloc( z(pp1:1:-1) /= 0.0_sp ))/p
END IF
                    ! Index lists for valid returns by date
ALLOCATE( tdx(p) )
DO i = 1,n
  tdx = Wher( r(i,:) /= -999.0_sp, cnt=m )
  ALLOCATE( sm(i)%idx(m) )
  sm(i)%idx = tdx(:m)
END DO
DEALLOCATE( tdx )
                    ! Sampling standard deviations and Lambda
                    ! noise estimates
CALL Sample( ln, r, std, w )
                    ! For k=0 there is no loading matrix, so start
                    ! with the sig's set to the sampling std
IF( k == 0 ) THEN
  z(:p) = std
  z(p+1:) = 0.0_sp
  v = Covar( z(:p) )
```

```
END IF
                    ! Loop on increasing # of factors
IF( prn /= 0 ) WRITE(*,'(a)') '   iter  steps    np     LLF' &
   & // '   LLF+np     gof   gof_ev  dgof/std'
kmn = k
IF( add_iter /= 0 ) THEN
  kmx = p
ELSE
  kmx = k
ENDIF
k = MAX( k - 1, 0 )
Factor_loop: DO WHILE( k < kmx )
                    ! Increase the # of factors by 1
  k = k + 1
  nz = p*k + p
                    ! Initialization & initial printout
  IF( k == kmn .OR. k == 1 ) THEN
    v = Covar( z(:nz) )
    gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
    nd = 0.5_sp*gof_ev
    iflag = 1
    f = Func( z(:nz) )
    np = COUNT( z(:nz) /= 0.0_sp )
    IF( prn /= 0 ) WRITE(*,'(3i8,2f10.3,3f10.1)') kmn, 0, np, f, f + np, &
       & gof, gof_ev, (gof - gof_ev)/gof_std
  END IF
                    ! Keep old values in case we have to back up
  f_old = f
  np_old = np
  z_old = z
                    ! Line search along the Lambda direction with
                    ! the biggest negative curvature (strongest
                    ! descent direction in the multivariate saddle
                    ! point); if there is no such direction exit
                    ! the factor loop
  IF( k > kmn ) THEN
    ALLOCATE( xi(nz) )
    xi(p*k+1:) = Next( z(:nz) )
    IF( ALL( xi(p*k+1:) == 0.0_sp ) ) EXIT
    xi(:p*k) = 0.0_sp
    v = Covar( z(:nz) )
    diagv = Get_diag( v )
    iflag = 2
    CALL Linmin( z(:nz), xi, f )
    DEALLOCATE( xi )
                    ! Cull Parameters
```

```
  z(p+1:nz) = Cull( ln, nsig, z(p+1:nz) )
  np = COUNT( z(:nz) /= 0.0_sp )
                    ! Rescale
  f = Func( z(:nz) )
  z(:p) = SQRT( norm/nd )*ABS( z(:p) )
END IF
                    ! The main optimization is done by calling the
                    ! nonlinear minimization program FRPRMN of
                    ! Numerical Recipes, as modified by A. Yahil.
                    ! Rescale the coavariance matrix for the
                    ! correct chi^2 before and after the
                    ! minimization.
iflag = 1
it = stepmx
CALL Frprmn( z(:nz), MAX( abstol, pp1/2*reltol ), 0.0_sp, it, f )
                    ! Cull Parameters
z(p+1:nz) = Cull( ln, nsig, z(p+1:nz) )
np = COUNT( z(:nz) /= 0.0_sp )
                    ! Rescale
f = Func( z(:nz) )
z(:p) = SQRT( norm/nd )*ABS( z(:p) )
                    ! Output
v = Covar( z(:nz) )
gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
f = Func( z(:nz) )
IF( prn /= 0 ) WRITE(*,'(3i8,2f10.3,3f10.1)') k, it, np, f, f + np, gof, &
   & gof_ev, (gof - gof_ev)/gof_std
                    ! Run FRPRMN again with the culled Lambda's
                    ! fixed.
ALLOCATE( tdx(nz) )
tdx = Wher( z(:nz) == 0.0, cnt=ndx )
ALLOCATE( idx(ndx) )
idx = tdx(:ndx)
it = stepmx
CALL Frprmn( z(:nz), MAX(abstol,pp1/2*reltol), 0.0_sp, it, f )
DEALLOCATE( idx, tdx )
                    ! Rescale
z(:p) = SQRT( norm/nd )*ABS( z(:p) )
v = Covar( z(:nz) )
gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
f = Func( z(:nz) )
IF( prn /= 0 ) WRITE(*,'(3i8,2f10.3,3f10.1)') k, it, np, f, f + np, gof, &
   & gof_ev, (gof - gof_ev)/gof_std
                    ! Back up to the previous solution if the
                    ! improvement in LLF is not significant at the
                    ! prescribed significance level, or there is
```

```
                    ! no increase in the # of parameters
  IF( k > kmn .AND. ( np <= np_old .OR. f >= f_old &
     & .OR. Gammq( 0.5_sp*ABS(np - np_old + EPSILON(1.0_sp)), &
     & 0.5_sp*ABS(f_old - f) ) > signif ) ) THEN
    IF( prn /= 0 ) WRITE(*,*) &
       'The last iteration was rejected as insignificant'
    k = k - 1
    nz = nz - p
    z(:nz) = z_old(:nz)
    z(nz+1:) = 0.0_sp
    v = Covar( z(:nz) )
    gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
    np = COUNT( z(:nz) /= 0.0_sp )
    EXIT
  END IF
 END DO Factor_loop
                    ! Encrypt nonzero z's
 WHERE( z /= 0.0_sp )
   z = SIGN( LOG( iz + ABS( z ) ), z )
 END WHERE
                    ! Cleanup
 DEALLOCATE( diagv )

END SUBROUTINE Algebron
```

```
/* ****************************************
/*  Copyright (C) 1998 by Algebron LLC.
/*  All rights reserved.
/*  Unauthorized reproduction prohibited.
/* **************************************** */
/* Time-stamp: <98/07/24 07:52:46 ayahil> */
/*

 Estimates the covariance matrix V(P,P) using the Algebron (TM) method which
 minimizes the complexity of V. See the variable list for specifics. Note
 that the input/output variable Z is set to zero for computations from
 scratch; otherwise the result of the previous computation needs to be
 provided.

 */

#ifndef ALGEBRON_H
#define ALGEBRON_H

void algebron_(
        double* abstol,  /* Absolute convergence criterion in
                    Maximum-likelihood minimization.  Input
                    [default=0.01] */
          int* add_iter,      /* Flag to add iterations.  Zero: do not add,
                    nonzero: add.  Input */
        double* chi2,    /* Chi-squared of sampled variables as a
                    function of time.  Output array(n) */
        double* gof,     /* Goodness-of-fit estimaor.  Output */
        double* gof_ev,  /* Expectation Value of GOF estimator.
                    Output */
        double* gof_std, /* Standard deviation of GOF estimator.
                    Output */
        int* n,          /* Total # of security samplings.  Input */
        double* nsig,    /* Number of sigma's below which a parameter is
                    culled.  Input [default=3]*/
        int* p,          /* Number of valid returns.  Input */
        int* prn,        /* Print flag.  Zero: do not print, nonzero:
                    print.  Input [default=1] */
          double* prob,      /* Probabilities of the chi-squared array.
                    Output array(n) */
        double* r,       /* valid returns.  Input array(n,p) */
        double* reltol,  /* Relative (to p*(p+1)/2) convergence
                    criterion in maximum-likelihood
                    minimization.  Input [default=0.0001] */
        double* signif,  /* Significance level at which to accept a new
                    iteration.  Input [default=0.01] */
```

```
        int* stepmx,    /* Maximum number of computation steps per
                        iteration.  Input [default=100] */
        double* v,      /* Estimated covariance matrix.  Output
                        array(p,p) */
        double* w,      /* Time weighting of the securities in the
                        computation of the covariance matrix.  Input
                        array(n) */
        double* z       /* Update parameters.  Input/Output
                        array(p*(p+1)) */
        );

double abstol_def = 0.01;

int add_iter_def = 1;

double nsig_def = 3.0;

int prn_def = 1;

double reltol_def = 0.0001;

double signif_def = 0.01;

int stepmx_def = 100;

#endif
```

```
/* ****************************************
/*  Copyright (C) 1998 by Algebron LLC.
/*  All rights reserved.
/*  Unauthorized reproduction prohibited.
/* **************************************** */
        /* Time-stamp: <98/07/08 09:36:43 ayahil> */
/*
 Mother of all ALGEBRON includes
 */

#ifndef ALGEBRON_ALL_H
#define ALGEBRON_ALL_H

#include "algebron.h"
#include "bootstrap.h"
#include "gen_covar.h"
#include "gen_dev.h"
#include "prepare.h"

#endif
```

```
                !****************************************
                ! Copyright (C) 1998 by Algebron LLC.
                ! All rights reserved.
                ! Unauthorized reproduction prohibited.
                !****************************************
                ! Time-stamp: <98/06/29 14:26:17 ayahil>
SUBROUTINE Algebron( &
  & abstol, &          ! Absolute error limit in minimization
  & add_iter, &        ! Flag for adding iterations
  & chi2, &           ! Chi^2 at each sampling point
  & gof, &            ! Goodness-of-fit measure
  & gof_ev, &          ! Expectation value of the GOF measure
  & gof_std, &         ! Standard deviation of the GOF measure
  & n_t, &            ! # of samples
  & nsig, &           ! # of sigma for Lambda truncation
  & p_t, &            ! # of valid securities
  & prn, &            ! Print flag
  & prob, &           ! Probabilities of chi^2 array
  & r_t, &            ! Returns
  & reltol, &         ! Relative error limit in minimization
  & signif, &         ! Significance level for accepting an iteration
  & stepmx, &          ! Maximum # of steps in FRPRMN
  & v, &              ! Resultant covariance matrix
  & w_t, &            ! Weights
  & z &               ! Update parameters
  & )

  USE Interfaces, ONLY: Covar, Cull, Func, Get_gof, Next, Sample
  USE Mynr, ONLY: Frprmn, Linmin
  USE Nr, ONLY: Gammq
  USE Nrtype
  USE Nrutil, ONLY: Assert_eq, Get_diag, Ifirstloc
  USE Parm, ONLY: diagv, idx, iflag, k, n, norm, p, r, sm, w
  USE Utils, ONLY: Permute, Wher
  IMPLICIT NONE
                ! Arguments
  INTEGER, INTENT(in) :: add_iter, prn, stepmx
  INTEGER, INTENT(in), TARGET :: n_t, p_t
  REAL(sp), INTENT(in) :: abstol, nsig, reltol, signif
  REAL(sp), INTENT(in), TARGET, DIMENSION(n_t) :: w_t
  REAL(sp), INTENT(in), TARGET, DIMENSION(n_t,p_t) :: r_t
  REAL(sp), INTENT(inout), DIMENSION(p_t*(p_t+1)) :: z
  REAL(sp), INTENT(out) :: gof, gof_ev, gof_std
  REAL(sp), INTENT(out), DIMENSION(n_t) :: chi2, prob
  REAL(sp), INTENT(out), DIMENSION(p_t,p_t) :: v
                ! Locals
```

```
 INTEGER :: chk, i, it, kmn, kmx, ndx, m, np, nz, pp1
 INTEGER, ALLOCATABLE, DIMENSION(:) :: tdx
 INTEGER, DIMENSION(p_t*(p_t+1)) :: iz
 LOGICAL, SAVE :: license_verified=.FALSE.
 REAL(sp) :: f, f_old, nd, np_old
 REAL(sp), ALLOCATABLE, DIMENSION(:) :: xi
 REAL(sp), DIMENSION(p_t*(p_t+1)) :: z_old
 REAL(sp), DIMENSION(p_t) :: ln, std
                    ! Externals
 INTEGER Vlslicense
                    ! Check for valid license
 IF( .NOT. license_verified ) THEN
   IF ( Vlslicense(  "LibAlgebron" // CHAR(0), "0" // CHAR(0) ) == 1) THEN
     license_verified = .TRUE.
     WRITE(*,*) 'Valid license'
   ELSE
     WRITE(*,*) 'No Valid license: cannot execute the ALGEBRON routine.'
!!$     RETURN
   ENDIF
 END IF
                    ! Initialization & pointers
 n => n_t
 p => p_t
 r => r_t
 w => w_t
 IF( ALLOCATED( sm ) ) DEALLOCATE( sm )
 ALLOCATE( diagv(p), sm(n) )
                    ! Check sizes
 chk = Assert_eq( (/ n, SIZE(chi2), SIZE(prob), SIZE(r,1), SIZE(w) /), &
    & ' Algebron-n' )
 chk = Assert_eq( p, SIZE(r,2), SIZE(v,1), SIZE(v,2) , ' Algebron-p' )
 pp1 = Assert_eq( p*(p+1), SIZE(z), ' Algebron-z' )
                    ! Decrypt input z & determine # of factors
 iz = Permute( pp1, seed=123571113 )
 IF( ALL( z == 0.0_sp ) ) THEN
   k = 0
 ELSE
   WHERE( z /= 0.0_sp )
     z = SIGN( EXP( ABS( z ) ) - iz, z )
   END WHERE
   k = (pp1 - Ifirstloc( z(pp1:1:-1) /= 0.0_sp ))/p
 END IF
                    ! Index lists for valid returns by date
 ALLOCATE( tdx(p) )
 DO i = 1,n
   tdx = Wher( r(i,:) /= -999.0_sp, cnt=m )
```

```
    ALLOCATE( sm(i)%idx(m) )
    sm(i)%idx = tdx(:m)
  END DO
  DEALLOCATE( tdx )
                    ! Sampling standard deviations and Lambda
                    ! noise estimates
  CALL Sample( ln, r, std, w )
                    ! For k=0 there is no loading matrix, so start
                    ! with the sig's set to the sampling std
  IF( k == 0 ) THEN
    z(:p) = std
    z(p+1:) = 0.0_sp
    v = Covar( z(:p) )
  END IF
                    ! Loop on increasing # of factors
  IF( prn /= 0 ) WRITE(*,'(a)') '   iter  steps     np      LLF' &
     & // '   LLF+np      gof    gof_ev  dgof/std'
  kmn = k
  IF( add_iter /= 0 ) THEN
    kmx = p
  ELSE
    kmx = k
  ENDIF
  k = MAX( k - 1, 0 )
  Factor_loop: DO WHILE( k < kmx )
                    ! Increase the # of factors by 1
    k = k + 1
    nz = p*k + p
                    ! Initialization & initial printout
    IF( k == kmn .OR. k == 1 ) THEN
      v = Covar( z(:nz) )
      gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
      nd = 0.5_sp*gof_ev
      iflag = 1
      f = Func( z(:nz) )
      np = COUNT( z(:nz) /= 0.0_sp )
      IF( prn /= 0 ) WRITE(*,'(3i8,2f10.3,3f10.1)') kmn, 0, np, f, f + np, &
         & gof, gof_ev, (gof - gof_ev)/gof_std
    END IF
                    ! Keep old values in case we have to back up
    f_old = f
    np_old = np
    z_old = z
                    ! Line search along the Lambda direction with
                    ! the biggest negative curvature (strongest
                    ! descent direction in the multivariate saddle
```

```
                    ! point); if there is no such direction exit
                    ! the factor loop
  IF( k > kmn ) THEN
    ALLOCATE( xi(nz) )
    xi(p*k+1:) = Next( z(:nz) )
    IF( ALL( xi(p*k+1:) == 0.0_sp ) ) EXIT
    xi(:p*k) = 0.0_sp
    v = Covar( z(:nz) )
    diagv = Get_diag( v )
    iflag = 2
    CALL Linmin( z(:nz), xi, f )
    DEALLOCATE( xi )
                    ! Cull Parameters
    z(p+1:nz) = Cull( ln, nsig, z(p+1:nz) )
    np = COUNT( z(:nz) /= 0.0_sp )
                    ! Rescale
    f = Func( z(:nz) )
    z(:p) = SQRT( norm/nd )*ABS( z(:p) )
  END IF
                    ! The main optimization is done by calling the
                    ! nonlinear minimization program FRPRMN of
                    ! Numerical Recipes, as modified by A. Yahil.
                    ! Rescale the coavariance matrix for the
                    ! correct chi^2 before and after the
                    ! minimization.
  iflag = 1
  it = stepmx
  CALL Frprmn( z(:nz), MAX( abstol, pp1/2*reltol ), 0.0_sp, it, f )
                    ! Cull Parameters
  z(p+1:nz) = Cull( ln, nsig, z(p+1:nz) )
  np = COUNT( z(:nz) /= 0.0_sp )
                    ! Rescale
  f = Func( z(:nz) )
  z(:p) = SQRT( norm/nd )*ABS( z(:p) )
                    ! Output
  v = Covar( z(:nz) )
  gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
  f = Func( z(:nz) )
  IF( prn /= 0 ) WRITE(*,'(3i8,2f10.3,3f10.1)') k, it, np, f, f + np, gof, &
     & gof_ev, (gof - gof_ev)/gof_std
                    ! Run FRPRMN again with the culled Lambda's
                    ! fixed.
  ALLOCATE( tdx(nz) )
  tdx = Wher( z(:nz) == 0.0, cnt=ndx )
  ALLOCATE( idx(ndx) )
  idx = tdx(:ndx)
```

```
    it = stepmx
    CALL Frprmn( z(:nz), MAX(abstol,pp1/2*reltol), 0.0_sp, it, f )
    DEALLOCATE( idx, tdx )
                        ! Rescale
    z(:p) = SQRT( norm/nd )*ABS( z(:p) )
    v = Covar( z(:nz) )
    gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
    f = Func( z(:nz) )
    IF( prn /= 0 ) WRITE(*,'(3i8,2f10.3,3f10.1)') k, it, np, f, f + np, gof, &
       & gof_ev, (gof - gof_ev)/gof_std
                        ! Back up to the previous solution if the
                        ! improvement in LLF is not significant at the
                        ! prescribed significance level, or there is
                        ! no increase in the # of parameters
    IF( k > kmn .AND. ( np <= np_old .OR. f >= f_old &
       & .OR. Gammq( 0.5_sp*ABS(np - np_old + EPSILON(1.0_sp)), &
       & 0.5_sp*ABS(f_old - f) ) > signif ) ) THEN
      IF( prn /= 0 ) WRITE(*,*) &
         'The last iteration was rejected as insignificant'
      k = k - 1
      nz = nz - p
      z(:nz) = z_old(:nz)
      z(nz+1:) = 0.0_sp
      v = Covar( z(:nz) )
      gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
      np = COUNT( z(:nz) /= 0.0_sp )
      EXIT
    END IF
  END DO Factor_loop
                        ! Encrypt nonzero z's
  WHERE( z /= 0.0_sp )
    z = SIGN( LOG( iz + ABS( z ) ), z )
  END WHERE
                        ! Cleanup
  DEALLOCATE( diagv )

END SUBROUTINE Algebron
```

```
                    !****************************************
                    ! Copyright (C) 1998 by Algebron LLC.
                    ! All rights reserved.
                    ! Unauthorized reproduction prohibited.
                    !****************************************
                    ! Time-stamp: <98/07/24 07:57:09 ayahil>
!!! Generate a bootstrap frequency distribution (with repetitions).

SUBROUTINE Bootstrap( &
   & freq, &              ! Bootstrap frequency distribution
   & n, &                 ! # of bootstrap samples
   & seed &               ! Initial seed for random number generator
   & )

 USE Dfport, ONLY: Time
 USE Nr, ONLY: Ran1
 USE Nrtype
 USE Ran_state, ONLY: Ran_seed
 IMPLICIT NONE
                    ! Arguments
 INTEGER, INTENT(in) :: n, seed
 INTEGER, INTENT(out), DIMENSION(n) :: freq
                    ! Locals
 INTEGER :: i
 INTEGER, DIMENSION(n) :: j
 LOGICAL, SAVE :: firsttime=.TRUE.
 REAL(sp), DIMENSION(n) :: t
                    ! Set the random number seed in the first call
                    ! to this subroutine-- 0: use Time(),
                    ! otherwise: use given seed.
 IF( firsttime ) THEN
   firsttime = .FALSE.
   IF( seed == 0 ) THEN
     CALL Ran_seed( Time() )
   ELSE
     CALL Ran_seed( seed )
   END IF
 END IF
                    ! Random frequency
 CALL Ran1( t )
 j = INT( t*n ) + 1
 freq = 0
 DO i = 1,n
   freq(j(i)) = freq(j(i)) + 1
 END DO
END
```

[bootstrap.f90]

```
                    /* ****************************************
                    /*  Copyright (C) 1998 by Algebron LLC.
                    /*  All rights reserved.
                    /*  Unauthorized reproduction prohibited.
                    /* **************************************** */
                            /* Time-stamp: <98/07/24 07:57:28 ayahil> */
/*

  Generate a bootstrap frequency distribution (with repetitions).

 */

#ifndef BOOTSTRAP_H
#define BOOTSTRAP_H

void bootstrap_(
        int* freq,   /* Bootstrap frequency distribution.  Output
                        array(n) */
        int* n,      /* Total # of samples.  Input */
        int* seed    /* Initial seed for random number generator.
                        Zero: use clock, otherwise: use input seed.
                        Input [default=0] */
        );

int seed_def = 0;

#endif
```

```
!*******************************************
! Copyright (C) 1997 by Amos Yahil.
! All Rights Reserved.
! Based on (C) Numerical Recipes software.
!*******************************************
! Time-stamp: <97/03/28 13:03:19 ayahil>
FUNCTION brent(ax,bx,cx,func,tol,xmin)
 USE nrtype; USE nrutil, ONLY : nrerror
 IMPLICIT NONE
 REAL(SP), INTENT(IN) :: ax,bx,cx,tol
 REAL(SP), INTENT(OUT) :: xmin
 REAL(SP) :: brent
 INTERFACE
   FUNCTION func(x,dx)
    USE nrtype
    IMPLICIT NONE
    REAL(SP), INTENT(IN) :: x
    REAL(SP), OPTIONAL, INTENT(OUT) :: dx
    REAL(SP) :: func
   END FUNCTION func
 END INTERFACE
 INTEGER(I4B), PARAMETER :: ITMAX=100
 REAL(SP), PARAMETER :: CGOLD=0.3819660_sp,ZEPS=1.0e-3_sp*EPSILON(ax)
 INTEGER(I4B) :: iter
 REAL(SP) :: a,b,d,e,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm
 a=MIN(ax,cx)
 b=MAX(ax,cx)
 v=bx
 w=v
 x=v
 e=0.0
 fx=func(x)
 fv=fx
 fw=fx
 DO iter=1,ITMAX
   xm=0.5_sp*(a+b)
   tol1=tol*ABS(x)+ZEPS
   tol2=2.0_sp*tol1
   IF (ABS(x-xm) <= (tol2-0.5_sp*(b-a))) THEN
     xmin=x
     brent=fx
     RETURN
   END IF
   IF (ABS(e) > tol1) THEN
     r=(x-w)*(fx-fv)
     q=(x-v)*(fx-fw)
```

```
      p=(x-v)*q-(x-w)*r
      q=2.0_sp*(q-r)
      IF (q > 0.0) p=-p
      q=ABS(q)
      etemp=e
      e=d
      IF (ABS(p) >= ABS(0.5_sp*q*etemp) .OR. &
          p <= q*(a-x) .OR. p >= q*(b-x)) THEN
        e=MERGE(a-x,b-x, x >= xm )
        d=CGOLD*e
      ELSE
        d=p/q
        u=x+d
        IF (u-a < tol2 .OR. b-u < tol2) d=SIGN(tol1,xm-x)
      END IF
    ELSE
      e=MERGE(a-x,b-x, x >= xm )
      d=CGOLD*e
    END IF
    u=MERGE(x+d,x+SIGN(tol1,d), ABS(d) >= tol1 )
    fu=func(u)
    IF (fu <= fx) THEN
      IF (u >= x) THEN
        a=x
      ELSE
        b=x
      END IF
      CALL shft(v,w,x,u)
      CALL shft(fv,fw,fx,fu)
    ELSE
      IF (u < x) THEN
        a=u
      ELSE
        b=u
      END IF
      IF (fu <= fw .OR. w == x) THEN
        v=w
        fv=fw
        w=u
        fw=fu
      ELSE IF (fu <= fv .OR. v == x .OR. v == w) THEN
        v=u
        fv=fu
      END IF
    END IF
  END DO
```

[brent.f90]

```
  CALL nrerror('brent: exceed maximum iterations')
CONTAINS
                        !BL
  SUBROUTINE shft(a,b,c,d)
    REAL(SP), INTENT(OUT) :: a
    REAL(SP), INTENT(INOUT) :: b,c
    REAL(SP), INTENT(IN) :: d
    a=b
    b=c
    c=d
  END SUBROUTINE shft
END FUNCTION brent
```

```
SUBROUTINE choldc(a,p)
USE nrtype; USE nrutil, ONLY : assert_eq,nrerror
IMPLICIT NONE
REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
REAL(SP), DIMENSION(:), INTENT(OUT) :: p
INTEGER(I4B) :: i,n
REAL(SP) :: summ
n=assert_eq(size(a,1),size(a,2),size(p),'choldc')
do i=1,n
        summ=a(i,i)-dot_product(a(i,1:i-1),a(i,1:i-1))
        if (summ <= 0.0) call nrerror('choldc failed')
        p(i)=sqrt(summ)
        a(i+1:n,i)=(a(i,i+1:n)-matmul(a(i+1:n,1:i-1),a(i,1:i-1)))/p(i)
end do
END SUBROUTINE choldc
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/26 21:17:31 ayahil>

!!! Inverse of a symmetric, positive-definite, matrix using the Cholesky
!!! decomposition. This routine assumes that the decomposition has been
!!! performed by Numerical Recipes routine CHOLDC, whose output is used for
!!! the matrix inversion. The output variables are both optional: ainv
!!! returns the full inverse matrix, diag only the diagonal part.

SUBROUTINE Cholinv( &
  & a, &              ! Input matrix as outputted by CHOLDC
  & p, &              ! Input diagonal as outputted by CHOLDC
  & ainv, &           ! Resultant inverse matrix
  & diag &            ! Resultant diagonal of inverse matrix
  & )

 USE Nrtype
 USE Nrutil, ONLY : Assert_eq, Get_diag
                     ! Arguments
 IMPLICIT NONE
 REAL(sp), INTENT(in), DIMENSION(:,:) :: a
 REAL(sp), INTENT(in), DIMENSION(:) :: p
 REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(a,1),SIZE(a,2)) :: ainv
 REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(a,1)) :: diag
                     ! Locals
 INTEGER :: i, j, n
 REAL(sp), DIMENSION(SIZE(a,1)) :: x
                     ! Check sizes
 n = Assert_eq( SIZE(a,1), SIZE(a,2), SIZE(p), ' Cholinv' )
 IF( PRESENT(ainv) ) THEN
   n = Assert_eq( n, SIZE(ainv,1), SIZE(ainv,2), ' Cholinv' )
 END IF
 IF( PRESENT(diag) ) THEN
   n = Assert_eq( n, SIZE(diag), ' Cholinv' )
 END IF
                     ! Matrix inversion
 IF( PRESENT(ainv) ) THEN
   DO j = 1,n
     x(j) = 1.0_sp/p(j)
     DO i = j+1,n
       x(i) = - DOT_PRODUCT(a(i,j:i-1),x(j:i-1))/p(i)
     END DO
```

```
      DO i = n,j,-1
        x(i) = (x(i) - DOT_PRODUCT(a(i+1:,i),x(i+1:)))/p(i)
      END DO
      DO i = j-1,1,-1
        x(i) = - DOT_PRODUCT(a(i+1:,i),x(i+1:))/p(i)
      END DO
      ainv(:,j) = x
    END DO
    IF( PRESENT(diag) ) THEN
      diag = Get_diag( ainv )
    END IF
  ELSE IF( PRESENT(diag) ) THEN
    DO j = 1,n
      x(j) = 1.0_sp/p(j)
      DO i = j+1,n
        x(i) = - DOT_PRODUCT(a(i,j:i-1),x(j:i-1))/p(i)
      END DO
      DO i = n,j,-1
        x(i) = (x(i) - DOT_PRODUCT(a(i+1:,i),x(i+1:)))/p(i)
      END DO
      diag(j) = x(j)
    END DO
  END IF

END SUBROUTINE Cholinv
```

```
SUBROUTINE cholsl(a,p,b,x)
USE nrtype; USE nrutil, ONLY : assert_eq
IMPLICIT NONE
REAL(SP), DIMENSION(:,:), INTENT(IN) :: a
REAL(SP), DIMENSION(:), INTENT(IN) :: p,b
REAL(SP), DIMENSION(:), INTENT(INOUT) :: x
INTEGER(I4B) :: i,n
n=assert_eq((/size(a,1),size(a,2),size(p),size(b),size(x)/),'cholsl')
do i=1,n
        x(i)=(b(i)-dot_product(a(i,1:i-1),x(1:i-1)))/p(i)
end do
do i=n,1,-1
        x(i)=(x(i)-dot_product(a(i+1:n,i),x(i+1:n)))/p(i)
end do
END SUBROUTINE cholsl
```

[cholsl.f90]

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/26 21:11:34 ayahil>

!!! Log-likelihood function and its gradient wrt to sig & lambda for factor
!!! analysis with missing data.  Called by the minimization routine Frprmn.
!!! NOTE: In this routine V is the *reduced* covariance matrix.

FUNCTION Cml( &
  & lambda, &          ! Loading matrix
  & sig, &           ! Independent standard deviations
  & dlambda &          ! Derivative of function wrt lambda
  & ) RESULT( out )

 USE Nrtype
 USE Nrutil, ONLY: Assert_eq, Diagadd, Get_diag, Outerprod
 USE Parm, ONLY: diagv, k, n, p, r, sm, w
 USE Utils, ONLY: Dmatmul_l, Dmatmul_r, Spd
 IMPLICIT NONE
                    ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:) :: sig
 REAL(sp), INTENT(in), DIMENSION(:,:) :: lambda
 REAL(sp), OPTIONAL, INTENT(out), DIMENSION(p,k) :: dlambda
 REAL(sp) :: out
                    ! Locals
 INTEGER :: chk, i, m
 INTEGER, DIMENSION(:), POINTER :: idx
 REAL(sp) :: dnorm, ldet
 REAL(sp), ALLOCATABLE, DIMENSION(:) :: b, x
 REAL(sp), ALLOCATABLE, DIMENSION(:,:) :: vinv
 REAL(sp), DIMENSION(p) :: dsig
 REAL(sp), DIMENSION(p,p) :: v
                    ! Check sizes
 chk = Assert_eq( k, SIZE(lambda,2), ' Cml-k' )
 chk = Assert_eq( n, SIZE(r,1), SIZE(sm), SIZE(w), ' Cml-n' )
 chk = Assert_eq( (/ p, SIZE(lambda,1), SIZE(r,2), SIZE(sig), SIZE(v,1), &
    & SIZE(v,2) /), ' Cml-p' )
 IF( PRESENT( dlambda) ) THEN
   chk = Assert_eq( k, SIZE(dlambda,2), ' Cml-k' )
   chk = Assert_eq( p, SIZE(dlambda,1), ' Cml-p' )
 END IF
                    ! Initialization
 out = 0.0_sp
```

```
  IF( PRESENT(dlambda) ) THEN
    dsig = 0.0_sp
    dlambda = 0.0_sp
  END IF
  v = MATMUL( lambda, TRANSPOSE(lambda) )
  CALL Diagadd( v, 1.0_sp )
                        ! Log-likelihood function
  Data_loop: DO i = 1,n
    idx => sm(i)%idx
    m = SIZE(idx)
    IF( m > 0 ) THEN
      ALLOCATE( b(m), x(m) )
      b = r(i,idx)/sig(idx)
      IF( PRESENT(dlambda) ) THEN
        ALLOCATE( vinv(m,m) )
        CALL Spd( v(idx,idx), ainv=vinv, b=b, ldet=ldet, x=x )
        dlambda(idx,:) = dlambda(idx,:) &
           & + w(i)*MATMUL( vinv - Outerprod( x, x ), lambda(idx,:) )
        dsig(idx) = dsig(idx) + w(i)*(1.0_sp - b*x)
        DEALLOCATE( vinv )
      ELSE
        CALL Spd( v(idx,idx), b=b, ldet=ldet, x=x )
      END IF
      dnorm = DOT_PRODUCT( b, x )
      DEALLOCATE( b, x )
      out = out + w(i)*(ldet + dnorm + SUM( LOG( sig(idx)**2 ) ))
    END IF
  END DO Data_loop
                        ! Optional gradient of likelihood function,
                        ! including penalty function
  IF( PRESENT(dlambda) ) THEN
    dlambda = 2.0_sp*(dlambda - lambda*(SPREAD( dsig/(1.0_sp &
       & + SUM( lambda**2, DIM=2 )), DIM=2, NCOPIES=k )))
  END IF

END FUNCTION Cml
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/22 06:33:30 ayahil>

!!! Covariance matrix from its factor representation.  The variables are
!!! passed as a 1-d array, for reasons of convenience in ALGEBRON.

FUNCTION Covar( &
   & z &                ! Minimization array
   & ) RESULT( out )

 USE Nrtype
 USE Nrutil, ONLY: Assert_eq, Diagadd
 USE Parm, ONLY: k, p
 USE Utils, ONLY: Dmatmul_l, Dmatmul_r
 IMPLICIT NONE
                      ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:) :: z
 REAL(sp), DIMENSION(p,p) :: out
                      ! Locals
 INTEGER :: chk
 REAL(sp), DIMENSION(p) :: sig
 REAL(sp), DIMENSION(p,k) :: lambda
                      ! Check sizes
 chk = Assert_eq( p*(k+1), SIZE(z), ' Covar-z' )
                      ! Initialization
 lambda = RESHAPE( z(p+1:), SHAPE(lambda) )
 sig = z(:p)
                      ! Covariance matrix
 out = MATMUL( lambda, TRANSPOSE(lambda) )
 CALL Diagadd( out, 1.0_sp )
 out = Dmatmul_r( Dmatmul_l( sig, out ), sig )

END FUNCTION Covar
```

```
                    !****************************************
                    ! Copyright (C) 1998 by Algebron LLC.
                    ! All rights reserved.
                    ! Unauthorized reproduction prohibited.
                    !****************************************
                    ! Time-stamp: <98/06/26 17:03:03 ayahil>

!!! Cull the loading matrix by truncating all elements below NSIG sigma's.
!!! The loading matrix is passed as a 1-d array, for reasons of convenience in
!!! ALGEBRON.

FUNCTION Cull( &
   & ln, &              ! Effective # of sampling points
   & nsig, &            ! SNR at which to truncate
   & z &               ! Minimization variables as single 1-d array
   & ) RESULT( out )

 USE Nrtype
 USE Nrutil, ONLY: Assert_eq
 USE Parm, ONLY: k, p
 IMPLICIT NONE
                    ! Arguments
 REAL(sp), INTENT(in) :: nsig
 REAL(sp), INTENT(in), DIMENSION(:) :: z
 REAL(sp), INTENT(in), DIMENSION(:) :: ln
 REAL(sp), DIMENSION(p*k) :: out
                    ! Locals
 INTEGER :: chk
                    ! Check sizes
 chk = Assert_eq( p, SIZE(ln), ' Cull-p' )
 chk = Assert_eq( p*k, SIZE(z), ' Cull-z' )
                    ! Cull
 WHERE( ABS( z )*RESHAPE( SPREAD( ln, DIM=2, NCOPIES=k ), SHAPE(z) ) &
    & < nsig )
   out = 0.0_sp
 ELSEWHERE
   out = z
 END WHERE

END FUNCTION Cull
```

```
MODULE Dflib

 INTERFACE
   SUBROUTINE Getarg( n, buffer )
    INTEGER :: n
    CHARACTER*(*) :: buffer
   END SUBROUTINE Getarg
 END INTERFACE

END MODULE Dflib
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/10 19:19:02 ayahil>

!!! Holding place for Unix intrinsic functions ported over to Digital FORTRAN
!!! for NT.

MODULE Dfport

  INTEGER :: Iargc, Time
  REAL :: Dtime
  EXTERNAL Dtime, Iargc, Time

END MODULE Dfport
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/05/06 10:45:45 ayahil>

!!! Matrix multiplication when one of the matrices is a diagonal matrix.

FUNCTION Dmatmul_l( &
  & diag, &              ! Left diagonal matrix
  & mat &                ! Right full matrix
  & ) RESULT( out )

 USE Nrtype
 USE Nrutil, ONLY: Assert_eq
 IMPLICIT NONE
                        ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:) :: diag
 REAL(sp), INTENT(in), DIMENSION(:,:) :: mat
 REAL(sp), DIMENSION(SIZE(diag),SIZE(mat,2)) :: out
                        ! Locals
 INTEGER :: n
                        ! Check sizes
 n = Assert_eq( SIZE(diag), SIZE(mat,1), ' Diag_matmul_l' )
                        ! Product
 out = SPREAD(diag,2,SIZE(diag)) * mat

END FUNCTION Dmatmul_l

FUNCTION Dmatmul_r( &
  & mat, &               ! Left full matrix
  & diag &               ! Right diagonal matrix
  & ) RESULT( out )

 USE Nrtype
 USE Nrutil, ONLY: Assert_eq
 IMPLICIT NONE
                        ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:) :: diag
 REAL(sp), INTENT(in), DIMENSION(:,:) :: mat
 REAL(sp), DIMENSION(SIZE(mat,1),SIZE(diag)) :: out
                        ! Locals
 INTEGER :: n
                        ! Check sizes
 n = Assert_eq( SIZE(mat,2), SIZE(diag), ' Diag_matmul_r' )
```

```
                    ! Product
  out = mat * SPREAD(diag,1,SIZE(diag))

END FUNCTION Dmatmul_r
```

```
SUBROUTINE eigsrt(d,v)
USE nrtype; USE nrutil, ONLY : assert_eq,imaxloc,swap
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(INOUT) :: d
REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: v
INTEGER(I4B) :: i,j,n
n=assert_eq(size(d),size(v,1),size(v,2),'eigsrt')
do i=1,n-1
        j=imaxloc(d(i:n))+i-1
        if (j /= i) then
                call swap(d(i),d(j))
                call swap(v(:,i),v(:,j))
        end if
end do
END SUBROUTINE eigsrt
```

```
!*****************************************
! Copyright (C) 1997 by Amos Yahil.
! All Rights Reserved.
! Based on (C) Numerical Recipes software.
!*****************************************
! Time-stamp: <98/06/29 14:29:17 ayahil>

!!! Nonlinear minimization routine of Numerical Recipes with corrections by A.
!!! Yahil.

SUBROUTINE Frprmn( p, abstol, reltol, iter, fret, lop, hip, ipr )

 USE Mynr, ONLY: Linmin
 USE Nrtype; USE Nrutil, ONLY: Assert_eq, Nrerror
 IMPLICIT NONE
 INTEGER, INTENT(inout) :: iter
 INTEGER, OPTIONAL, INTENT(in) :: ipr
 REAL(sp), INTENT(in) :: abstol,reltol
 REAL(sp), INTENT(inout), DIMENSION(:) :: p
 REAL(sp), INTENT(out) :: fret
 REAL(sp), OPTIONAL, INTENT(in), DIMENSION(:) :: hip,lop
 INTERFACE
   FUNCTION func(p,xi)
    USE nrtype
    IMPLICIT NONE
    REAL(sp), INTENT(inout), DIMENSION(:) :: p
    REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(p)) :: xi
    REAL(sp) :: func
   END FUNCTION func
 END INTERFACE
 INTEGER :: i,itmax,its
 REAL(sp) :: dgg,dx,fp,gam,gg
 REAL(sp), DIMENSION(SIZE(p)) :: g,h,xi
                    !Check that limits are not exceeded
 IF( PRESENT(lop) ) THEN
   its = Assert_eq( SIZE(p), SIZE(lop), ' Frprmn-lop' )
   IF( MINVAL(p-lop) < 0.0 ) THEN
     WRITE(0,*) p
     CALL Nrerror( ' Frprmn: p < lop' )
   END IF
 END IF
 IF( PRESENT(hip) ) THEN
   its = Assert_eq( SIZE(p), SIZE(hip), ' Frprmn-hip' )
   IF( MAXVAL(p-hip) > 0.0 ) THEN
     WRITE(0,*) p
     CALL Nrerror( ' Frprmn: p > hip' )
```

```
  END IF
END IF
                    !Initial function and gradient
fret = Func( p, xi )
                    !If at limit, don't allow gradient to push out
IF( PRESENT(lop) ) THEN
  WHERE (p == lop .AND. xi > 0.0)
    xi = 0.0
  END WHERE
END IF
IF( PRESENT(hip) ) THEN
  WHERE (p == hip .AND. xi < 0.0)
    xi = 0.0
  END WHERE
END IF
                    !Initial conjugate gradient in the direction
                    ! of the gradient
g = -xi
h = g
xi = h
                    !Iteration loop
itmax = iter
iter = 1
IF( ALL(xi == 0.0) ) RETURN
Iteration_loop: DO WHILE(iter <= itmax)
                    !Save previous function value
  fp = fret
                    !Line minimization
  dx = 1.0_sp/SQRT( DOT_PRODUCT( xi, xi ) )
  IF( PRESENT(lop) ) THEN
    IF( PRESENT(hip) ) THEN
      CALL Linmin( p, xi, fret, dx=dx, lop=lop, hip=hip )
    ELSE
      CALL Linmin( p, xi, fret, dx=dx, lop=lop )
    END IF
  ELSE IF( PRESENT(hip) ) THEN
    CALL Linmin( p, xi, fret, dx=dx, hip=hip )
  ELSE
    CALL Linmin( p, xi, fret, dx=dx )
  END IF
                    !Optional print
  IF( PRESENT(ipr) ) THEN
    IF( ipr /= 0 ) THEN
      WRITE(0,'(a,i5,1p4e12.4)') 'iter, fret, gam, min(p), max(p)', &
        &                  iter, fret, gam, MINVAL(p), MAXVAL(p)
    END IF
```

```
  END IF
                    !Convergence test
  IF( ABS(fret-fp) <= abstol + 0.5*(reltol+EPSILON(1.0_sp)) &
     & *(ABS(fret)+ABS(fp)) ) RETURN
  fret = Func( p, xi )
                    !If at limit, don't allow gradient to push out
  IF( PRESENT(lop) ) THEN
    WHERE (p == lop .AND. xi > 0.)
      xi = 0.
      h = 0.
    END WHERE
  END IF
  IF( PRESENT(hip) ) THEN
    WHERE (p == hip .AND. xi < 0.)
      xi = 0.
      h = 0.
    END WHERE
  END IF
  IF( ALL(xi == 0.) ) RETURN
                    !Set the next conjugate gradient direction
  iter = iter + 1
  gg = DOT_PRODUCT(g,g)
  !!dgg = DOT_PRODUCT( xi, xi ) !Fletcher-Reeves
  dgg = DOT_PRODUCT( xi+g, xi ) !Polack-Ribiere
  gam = dgg/gg
  g = -xi
  h = g + gam*h
  xi = h
 END DO Iteration_loop
                    !No convergence (warning only)
 WRITE(0,*) 'Frprmn: maximum iterations exceeded',itmax

END SUBROUTINE frprmn
```

```
                    !****************************************
                    ! Copyright (C) 1998 by Algebron LLC.
                    ! All rights reserved.
                    ! Unauthorized reproduction prohibited.
                    !****************************************
                    ! Time-stamp: <98/06/29 14:07:45 ayahil>

!!! This routine is simply a traffic cop directing to the actual minimization
!!! functions, depending on IFLAG.  There is also an option to prevent
!!! minimization wrt to some variables by setting the appropriate gradient
!!! components to zero.

FUNCTION Func( &
  & z, &                ! Arguments of minimization function
  & xi &                ! Gradient of minimization function wrt z
  & ) RESULT( out )

 USE Interfaces, ONLY: Cml, Ml
 USE Nrtype
 USE Nrutil, ONLY: Assert_eq
 USE Parm, ONLY: diagv, idx, iflag, k, p, n
 IMPLICIT NONE
                    ! Arguments
 REAL(sp), INTENT(inout), TARGET, DIMENSION(:) :: z
 REAL(sp), OPTIONAL, INTENT(out), TARGET, DIMENSION(SIZE(z)) :: xi
 REAL(sp) :: out
                    ! Locals
 INTEGER :: chk
 REAL(sp), DIMENSION(p,k) :: dlambda, lambda
 REAL(sp), POINTER, DIMENSION(:) :: dsig, sig
                    ! Check sizes
 IF( PRESENT(xi) ) THEN
   chk = Assert_eq( p*(k+1), SIZE(z), SIZE(xi), ' Func-z' )
 ELSE
   chk = Assert_eq( p*(k+1), SIZE(z), ' Func-z' )
 END IF
                    ! Initialization
 sig => z(:p)
 IF( PRESENT(xi) ) THEN
   dsig => xi(:p)
 END IF
 lambda = RESHAPE(z(p+1:),(/p,k/))
                    ! Select minimization function
 SELECT CASE( iflag )
                    ! Standard maximum likelihood
 CASE( 1 )
```

```
    IF( PRESENT(xi) ) THEN
      out = Ml( lambda, sig, dlambda=dlambda, dsig=dsig )
      xi(p+1:) = RESHAPE(dlambda,(/p*k/))
    ELSE
      out = Ml( lambda, sig )
    END IF
                        ! Maximum likelihood costrained to maintaining
                        ! the diagonal of the covariance matrix
                        ! constant
  CASE( 2 )
    sig = SQRT( diagv/(1.0_sp + SUM( lambda**2, DIM=2 )) )
    IF( PRESENT(xi) ) THEN
      dsig = 0.0_sp
      out = Cml( lambda, sig, dlambda=dlambda )
      xi(p+1:) = RESHAPE(dlambda,(/p*k/))
    ELSE
      out = Cml( lambda, sig )
    END IF
  CASE DEFAULT
    STOP 'Func: illegal iflag'
  END SELECT
                        ! Disallow gradients for variables marked to
                        ! be constant
  IF( PRESENT(xi) .AND. ALLOCATED(idx) ) THEN
    xi(idx) = 0.0_sp
  END IF

END FUNCTION Func
```

```
FUNCTION gammln_s(xx)
USE nrtype; USE nrutil, ONLY : arth,assert
IMPLICIT NONE
REAL(SP), INTENT(IN) :: xx
REAL(SP) :: gammln_s
REAL(DP) :: tmp,x
REAL(DP) :: stp = 2.5066282746310005_dp
REAL(DP), DIMENSION(6) :: coef = (/76.18009172947146_dp,&
        -86.50532032941677_dp,24.01409824083091_dp,&
        -1.231739572450155_dp,0.1208650973866179e-2_dp,&
        -0.5395239384953e-5_dp/)
call assert(xx > 0.0, 'gammln_s arg')
x=xx
tmp=x+5.5_dp
tmp=(x+0.5_dp)*log(tmp)-tmp
gammln_s=tmp+log(stp*(1.000000000190015_dp+&
        sum(coef(:)/arth(x+1.0_dp,1.0_dp,size(coef))))/x)
END FUNCTION gammln_s


FUNCTION gammln_v(xx)
USE nrtype; USE nrutil, ONLY: assert
IMPLICIT NONE
INTEGER(I4B) :: i
REAL(SP), DIMENSION(:), INTENT(IN) :: xx
REAL(SP), DIMENSION(size(xx)) :: gammln_v
REAL(DP), DIMENSION(size(xx)) :: ser,tmp,x,y
REAL(DP) :: stp = 2.5066282746310005_dp
REAL(DP), DIMENSION(6) :: coef = (/76.18009172947146_dp,&
        -86.50532032941677_dp,24.01409824083091_dp,&
        -1.231739572450155_dp,0.1208650973866179e-2_dp,&
        -0.5395239384953e-5_dp/)
if (size(xx) == 0) RETURN
call assert(all(xx > 0.0), 'gammln_v arg')
x=xx
tmp=x+5.5_dp
tmp=(x+0.5_dp)*log(tmp)-tmp
ser=1.000000000190015_dp
y=x
do i=1,size(coef)
        y=y+1.0_dp
        ser=ser+coef(i)/y
end do
gammln_v=tmp+log(stp*ser/x)
END FUNCTION gammln_v
```

[gammln.f90]

```
FUNCTION gammq_s(a,x)
USE nrtype; USE nrutil, ONLY : assert
USE nr, ONLY : gcf,gser
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,x
REAL(SP) :: gammq_s
call assert( x >= 0.0,  a > 0.0, 'gammq_s args')
if (x<a+1.0_sp) then
        gammq_s=1.0_sp-gser(a,x)
else
        gammq_s=gcf(a,x)
end if
END FUNCTION gammq_s


FUNCTION gammq_v(a,x)
USE nrtype; USE nrutil, ONLY : assert,assert_eq
USE nr, ONLY : gcf,gser
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
REAL(SP), DIMENSION(size(a)) :: gammq_v
LOGICAL(LGT), DIMENSION(size(x)) :: mask
INTEGER(I4B) :: ndum
ndum=assert_eq(size(a),size(x),'gammq_v')
call assert( all(x >= 0.0),  all(a > 0.0), 'gammq_v args')
mask = (x<a+1.0_sp)
gammq_v=merge(1.0_sp-gser(a,merge(x,0.0_sp,mask)), &
        gcf(a,merge(x,0.0_sp,.not. mask)),mask)
END FUNCTION gammq_v
```

```
SUBROUTINE gasdev_s(harvest)
USE nrtype
USE nr, ONLY : ran1
IMPLICIT NONE
REAL(SP), INTENT(OUT) :: harvest
REAL(SP) :: rsq,v1,v2
REAL(SP), SAVE :: g
LOGICAL, SAVE :: gaus_stored=.false.
if (gaus_stored) then
        harvest=g
        gaus_stored=.false.
else
        do
                call ran1(v1)
                call ran1(v2)
                v1=2.0_sp*v1-1.0_sp
                v2=2.0_sp*v2-1.0_sp
                rsq=v1**2+v2**2
                if (rsq > 0.0 .and. rsq < 1.0) exit
        end do
        rsq=sqrt(-2.0_sp*log(rsq)/rsq)
        harvest=v1*rsq
        g=v2*rsq
        gaus_stored=.true.
end if
END SUBROUTINE gasdev_s

SUBROUTINE gasdev_v(harvest)
USE nrtype; USE nrutil, ONLY : array_copy
USE nr, ONLY : ran1
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
REAL(SP), DIMENSION(size(harvest)) :: rsq,v1,v2
REAL(SP), ALLOCATABLE, DIMENSION(:), SAVE :: g
INTEGER(I4B) :: n,ng,nn,m
INTEGER(I4B), SAVE :: last_allocated=0
LOGICAL, SAVE :: gaus_stored=.false.
LOGICAL, DIMENSION(size(harvest)) :: mask
n=size(harvest)
if (n /= last_allocated) then
        if (last_allocated /= 0) deallocate(g)
        allocate(g(n))
        last_allocated=n
        gaus_stored=.false.
end if
if (gaus_stored) then
```

```
        harvest=g
        gaus_stored=.false.
    else
        ng=1
        do
            if (ng > n) exit
            call ran1(v1(ng:n))
            call ran1(v2(ng:n))
            v1(ng:n)=2.0_sp*v1(ng:n)-1.0_sp
            v2(ng:n)=2.0_sp*v2(ng:n)-1.0_sp
            rsq(ng:n)=v1(ng:n)**2+v2(ng:n)**2
            mask(ng:n)=(rsq(ng:n)>0.0 .and. rsq(ng:n)<1.0)
            call array_copy(pack(v1(ng:n),mask(ng:n)),v1(ng:),nn,m)
            v2(ng:ng+nn-1)=pack(v2(ng:n),mask(ng:n))
            rsq(ng:ng+nn-1)=pack(rsq(ng:n),mask(ng:n))
            ng=ng+nn
        end do
        rsq=sqrt(-2.0_sp*log(rsq)/rsq)
        harvest=v1*rsq
        g=v2*rsq
        gaus_stored=.true.
    end if
    END SUBROUTINE gasdev_v
```

```
FUNCTION gcf_s(a,x,gln)
USE nrtype; USE nrutil, ONLY : nrerror
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,x
REAL(SP), OPTIONAL, INTENT(OUT) :: gln
REAL(SP) :: gcf_s
INTEGER(I4B), PARAMETER :: ITMAX=100
REAL(SP), PARAMETER :: EPS=epsilon(x),FPMIN=tiny(x)/EPS
INTEGER(I4B) :: i
REAL(SP) :: an,b,c,d,del,h
if (x == 0.0) then
        gcf_s=1.0
        RETURN
end if
b=x+1.0_sp-a
c=1.0_sp/FPMIN
d=1.0_sp/b
h=d
do i=1,ITMAX
        an=-i*(i-a)
        b=b+2.0_sp
        d=an*d+b
        if (abs(d) < FPMIN) d=FPMIN
        c=b+an/c
        if (abs(c) < FPMIN) c=FPMIN
        d=1.0_sp/d
        del=d*c
        h=h*del
        if (abs(del-1.0_sp) <= EPS) exit
end do
if (i > ITMAX) call nrerror('a too large, ITMAX too small in gcf_s')
if (present(gln)) then
        gln=gammln(a)
        gcf_s=exp(-x+a*log(x)-gln)*h
else
        gcf_s=exp(-x+a*log(x)-gammln(a))*h
end if
END FUNCTION gcf_s


FUNCTION gcf_v(a,x,gln)
USE nrtype; USE nrutil, ONLY : assert_eq,nrerror
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
```

```
REAL(SP), DIMENSION(:), OPTIONAL, INTENT(OUT) :: gln
REAL(SP), DIMENSION(size(a)) :: gcf_v
INTEGER(I4B), PARAMETER :: ITMAX=100
REAL(SP), PARAMETER :: EPS=epsilon(x),FPMIN=tiny(x)/EPS
INTEGER(I4B) :: i
REAL(SP), DIMENSION(size(a)) :: an,b,c,d,del,h
LOGICAL(LGT), DIMENSION(size(a)) :: converged,zero
i=assert_eq(size(a),size(x),'gcf_v')
zero=(x == 0.0)
where (zero)
        gcf_v=1.0
elsewhere
        b=x+1.0_sp-a
        c=1.0_sp/FPMIN
        d=1.0_sp/b
        h=d
end where
converged=zero
do i=1,ITMAX
        where (.not. converged)
                an=-i*(i-a)
                b=b+2.0_sp
                d=an*d+b
                d=merge(FPMIN,d, abs(d)<FPMIN )
                c=b+an/c
                c=merge(FPMIN,c, abs(c)<FPMIN )
                d=1.0_sp/d
                del=d*c
                h=h*del
                converged = (abs(del-1.0_sp)<=EPS)
        end where
        if (all(converged)) exit
end do
if (i > ITMAX) call nrerror('a too large, ITMAX too small in gcf_v')
if (present(gln)) then
        if (size(gln) < size(a)) call &
                nrerror('gser: Not enough space for gln')
        gln=gammln(a)
        where (.not. zero) gcf_v=exp(-x+a*log(x)-gln)*h
else
        where (.not. zero) gcf_v=exp(-x+a*log(x)-gammln(a))*h
end if
END FUNCTION gcf_v
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/26 20:22:42 ayahil>

!!! Generate a random p x p covariance matrix

SUBROUTINE Gen_covar( &
  & c, &              ! Output covariance matrix
  & p &               ! Dimension of covariance matrix
  & )

 USE Nr, ONLY: Ran1
 USE Nrtype
 USE Nrutil, ONLY: Assert_eq, Outerprod, Unit_matrix
 IMPLICIT NONE
                     ! Arguments
 INTEGER, INTENT(in) :: p
 REAL(sp), INTENT(out), DIMENSION(p,p) :: c
                     ! Locals
 INTEGER :: chk, i
 REAL(sp), DIMENSION(p) :: e
 REAL(sp), DIMENSION(p,p) :: proj, r
                     ! Check sizes
 chk = Assert_eq( p, SIZE(c,1), SIZE(c,2), ' Covar-p' )
                     ! Generate random eigenvalues
 CALL Ran1( e )
                     ! Generate a random orthogonal transformation
 CALL Ran1( r(:,1) )
 r(:,1) = 2.0_sp*r(:,1) - 1.0_sp
 r(:,1) = r(:,1)/SQRT( SUM( r(:,1)**2 ) + TINY(1.0_sp) )
 CALL Unit_matrix( proj )
 DO i = 2,p
   proj = proj - Outerprod( r(:,i-1), r(:,i-1) )
   CALL Ran1( r(:,i) )
   r(:,i) = MATMUL( proj, 2.0_sp*r(:,i) - 1.0_sp )
   r(:,i) = r(:,i)/SQRT( SUM( r(:,i)**2 ) + TINY(1.0_sp))
 END DO
                     ! Rotate the eigenvalues
 DO i = 1,p
   c(:,i) = e(i)*r(:,i)
 END DO
 c = MATMUL( c, TRANSPOSE( r ) )
```

END SUBROUTINE Gen_covar

```
/* ****************************************
/*  Copyright (C) 1998 by Algebron LLC.
/*  All rights reserved.
/*  Unauthorized reproduction prohibited.
/* **************************************** */
        /* Time-stamp: <98/06/13 16:46:06 ayahil> */
/*

 Generate a random covariance matrix C(P,P).

 */

#ifndef GEN_COVAR_H
#define GEN_COVAR_H

void gen_covar(
        double* c,      /* Covariance matrix.  Output array(p,p) */
        int* p          /* # of variables.  Input */
        );

#endif
```

```
!*****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!*****************************************
! Time-stamp: <98/06/26 20:23:21 ayahil>

!!! Generate multivariate normal deviates with covariance matrix c.

SUBROUTINE Gen_dev( &
  & c, &              ! Covariance matrix
  & n, &              ! # of deviates
  & p, &              ! # of variables
  & x &               ! Array of deviates
  & )

 USE Nr, ONLY: Gasdev, Tqli, Tred2
 USE Nrtype
 USE Nrutil, ONLY: Assert_eq
 IMPLICIT NONE
                     ! Arguments
 INTEGER, INTENT(in) :: n, p
 REAL(sp), INTENT(in), DIMENSION(p,p) :: c
 REAL(sp), INTENT(out), DIMENSION(n,p) :: x
                     ! Locals
 INTEGER :: chk, j
 REAL(sp), DIMENSION(p) :: d, e
 REAL(sp), DIMENSION(p,p) :: a
                     ! Check sizes
 chk = Assert_eq( n, SIZE(x,1), ' Gen_dev-n' )
 chk = Assert_eq( p, SIZE(c,1), SIZE(c,2), SIZE(x,2), ' Gen_dev-p' )
                     ! Eigenvalues/vectors of the covariance matrix
 a = c
 CALL Tred2( a, d, e )
 CALL Tqli( d, e, a )
                     ! Random Gaussian deviates in eigenvector basis
 DO j = 1, p
   CALL Gasdev( x(:,j) )
   x(:,j) = SQRT( d(j) )*x(:,j)
 END DO
                     ! Rotate to given basis
 x = MATMUL( x, TRANSPOSE(a) )

END SUBROUTINE Gen_dev
```

```
                    /* ****************************************
                    /*  Copyright (C) 1998 by Algebron LLC.
                    /*  All rights reserved.
                    /*  Unauthorized reproduction prohibited.
                    /* **************************************** */
                            /* Time-stamp: <98/06/13 16:42:15 ayahil> */
/*

 Generate N multivariate normal deviates with a covariance matrix C(P,P).

 */

#ifndef GEN_DEV_H
#define GEN_DEV_H

void gen_dev_(
        double* c, /* Covariance matrix.  Input array(p,p) */
        int* n,          /* # of deviates.  Input */
        int* p,          /* # of variables.  Input */
        double* x        /* Random deviates generated.  Output
                            array(n,p) */
        );

#endif
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/26 21:13:26 ayahil>

!!! Goodness-of-fit measure

FUNCTION Get_gof( &
  & chi2, &          ! Chi^2 at each sampling point
  & gof_ev, &        ! Expectation value of GOF measure
  & gof_std, &       ! Standard deviation of GOF measure
  & prob, &          ! Probabilities of chi^2 array
  & r, &             ! Unbiased sampling matrix
  & v, &             ! Covariance matrix
  & w &              ! Weights
  & ) RESULT( out )

 USE Nr, ONLY: Gammq
 USE Nrtype
 USE Nrutil, ONLY: Assert_eq
 USE Parm, ONLY: n, p, sm
 USE Utils, ONLY: Spd
 IMPLICIT NONE
                    ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:) :: w
 REAL(sp), INTENT(in), DIMENSION(:,:) :: r, v
 REAL(sp), INTENT(out) :: gof_ev, gof_std
 REAL(sp), INTENT(out), DIMENSION(n) :: chi2, prob
 REAL(sp) :: out
                    ! Locals
 INTEGER :: chk, i, m
 INTEGER, POINTER, DIMENSION(:) :: idx
 REAL(sp), ALLOCATABLE, DIMENSION(:) :: x
                    ! Check sizes
 chk = Assert_eq( (/ n, SIZE(chi2), SIZE( prob), SIZE(r,1), SIZE(w) /), &
    & ' Gof-n' )
 chk = Assert_eq( p, SIZE(r,2), SIZE(v,1), SIZE(v,2), ' Gof-p' )
                    ! Build up 2*chi^2 a sampling point at a time
 gof_ev = 0.0_sp
 gof_std = 0.0_sp
 out = 0.0_sp
 Data_loop: DO i = 1,n
   idx => sm(i)%idx
   m = SIZE(idx)
```

```
    IF( m > 0 ) THEN
      ALLOCATE( x(m) )
      CALL Spd( v(idx,idx), b=r(i,idx), x=x )
      chi2(i) = DOT_PRODUCT( r(i,idx), x )
      out = out + w(i)*(chi2(i) - m)**2
      DEALLOCATE( x )
      gof_ev = gof_ev + 2*m*w(i)
      gof_std = gof_std + 8*m*(m + 6)*w(i)**2
      prob(i) = Gammq( 0.5_sp*m, 0.5_sp*chi2(i) )
    ELSE
      chi2(i) = 0.0_sp
      prob(i) = 0.0_sp
    END IF
  END DO Data_loop
  gof_std = SQRT( gof_std )

END FUNCTION Get_gof
```

```
FUNCTION gser_s(a,x,gln)
USE nrtype; USE nrutil, ONLY : nrerror
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,x
REAL(SP), OPTIONAL, INTENT(OUT) :: gln
REAL(SP) :: gser_s
INTEGER(I4B), PARAMETER :: ITMAX=100
REAL(SP), PARAMETER :: EPS=epsilon(x)
INTEGER(I4B) :: n
REAL(SP) :: ap,del,summ
if (x == 0.0) then
        gser_s=0.0
        RETURN
end if
ap=a
summ=1.0_sp/a
del=summ
do n=1,ITMAX
        ap=ap+1.0_sp
        del=del*x/ap
        summ=summ+del
        if (abs(del) < abs(summ)*EPS) exit
end do
if (n > ITMAX) call nrerror('a too large, ITMAX too small in gser_s')
if (present(gln)) then
        gln=gammln(a)
        gser_s=summ*exp(-x+a*log(x)-gln)
else
        gser_s=summ*exp(-x+a*log(x)-gammln(a))
end if
END FUNCTION gser_s


FUNCTION gser_v(a,x,gln)
USE nrtype; USE nrutil, ONLY : assert_eq,nrerror
USE nr, ONLY : gammln
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
REAL(SP), DIMENSION(:), OPTIONAL, INTENT(OUT) :: gln
REAL(SP), DIMENSION(size(a)) :: gser_v
INTEGER(I4B), PARAMETER :: ITMAX=100
REAL(SP), PARAMETER :: EPS=epsilon(x)
INTEGER(I4B) :: n
REAL(SP), DIMENSION(size(a)) :: ap,del,summ
LOGICAL(LGT), DIMENSION(size(a)) :: converged,zero
```

```
n=assert_eq(size(a),size(x),'gser_v')
zero=(x == 0.0)
where (zero) gser_v=0.0
ap=a
summ=1.0_sp/a
del=summ
converged=zero
do n=1,ITMAX
        where (.not. converged)
                ap=ap+1.0_sp
                del=del*x/ap
                summ=summ+del
                converged = (abs(del) < abs(summ)*EPS)
        end where
        if (all(converged)) exit
end do
if (n > ITMAX) call nrerror('a too large, ITMAX too small in gser_v')
if (present(gln)) then
        if (size(gln) < size(a)) call &
                nrerror('gser: Not enough space for gln')
        gln=gammln(a)
        where (.not. zero) gser_v=summ*exp(-x+a*log(x)-gln)
else
        where (.not. zero) gser_v=summ*exp(-x+a*log(x)-gammln(a))
end if
END FUNCTION gser_v
```

```
SUBROUTINE indexx_sp(arr,index)
USE nrtype; USE nrutil, ONLY : arth,assert_eq,nrerror,swap
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(IN) :: arr
INTEGER(I4B), DIMENSION(:), INTENT(OUT) :: index
INTEGER(I4B), PARAMETER :: NN=15, NSTACK=50
REAL(SP) :: a
INTEGER(I4B) :: n,k,i,j,indext,jstack,l,r
INTEGER(I4B), DIMENSION(NSTACK) :: istack
n=assert_eq(size(index),size(arr),'indexx_sp')
index=arth(1,1,n)
jstack=0
l=1
r=n
do
        if (r-l < NN) then
                do j=l+1,r
                        indext=index(j)
                        a=arr(indext)
                        do i=j-1,l,-1
                                if (arr(index(i)) <= a) exit
                                index(i+1)=index(i)
                        end do
                        index(i+1)=indext
                end do
                if (jstack == 0) RETURN
                r=istack(jstack)
                l=istack(jstack-1)
                jstack=jstack-2
        else
                k=(l+r)/2
                call swap(index(k),index(l+1))
                call icomp_xchg(index(l),index(r))
                call icomp_xchg(index(l+1),index(r))
                call icomp_xchg(index(l),index(l+1))
                i=l+1
                j=r
                indext=index(l+1)
                a=arr(indext)
                do
                        do
                                i=i+1
                                if (arr(index(i)) >= a) exit
                        end do
                        do
                                j=j-1
```

[indexx.f90]

```
                        if (arr(index(j)) <= a) exit
                    end do
                    if (j < i) exit
                    call swap(index(i),index(j))
                end do
                index(l+1)=index(j)
                index(j)=indext
                jstack=jstack+2
                if (jstack > NSTACK) call nrerror('indexx: NSTACK too small')
                if (r-i+1 >= j-l) then
                    istack(jstack)=r
                    istack(jstack-1)=i
                    r=j-1
                else
                    istack(jstack)=j-1
                    istack(jstack-1)=l
                    l=i
                end if
            end if
        end do
        CONTAINS
!BL
        SUBROUTINE icomp_xchg(i,j)
        INTEGER(I4B), INTENT(INOUT) :: i,j
        INTEGER(I4B) :: swp
        if (arr(j) < arr(i)) then
            swp=i
            i=j
            j=swp
        end if
        END SUBROUTINE icomp_xchg
        END SUBROUTINE indexx_sp

        SUBROUTINE indexx_i4b(iarr,index)
        USE nrtype; USE nrutil, ONLY : arth,assert_eq,nrerror,swap
        IMPLICIT NONE
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: iarr
        INTEGER(I4B), DIMENSION(:), INTENT(OUT) :: index
        INTEGER(I4B), PARAMETER :: NN=15, NSTACK=50
        INTEGER(I4B) :: a
        INTEGER(I4B) :: n,k,i,j,indext,jstack,l,r
        INTEGER(I4B), DIMENSION(NSTACK) :: istack
        n=assert_eq(size(index),size(iarr),'indexx_sp')
        index=arth(1,1,n)
        jstack=0
        l=1
```

```
r=n
do
        if (r-l < NN) then
                do j=l+1,r
                        indext=index(j)
                        a=iarr(indext)
                        do i=j-1,1,-1
                                if (iarr(index(i)) <= a) exit
                                index(i+1)=index(i)
                        end do
                        index(i+1)=indext
                end do
                if (jstack == 0) RETURN
                r=istack(jstack)
                l=istack(jstack-1)
                jstack=jstack-2
        else
                k=(l+r)/2
                call swap(index(k),index(l+1))
                call icomp_xchg(index(l),index(r))
                call icomp_xchg(index(l+1),index(r))
                call icomp_xchg(index(l),index(l+1))
                i=l+1
                j=r
                indext=index(l+1)
                a=iarr(indext)
                do
                        do
                                i=i+1
                                if (iarr(index(i)) >= a) exit
                        end do
                        do
                                j=j-1
                                if (iarr(index(j)) <= a) exit
                        end do
                        if (j < i) exit
                        call swap(index(i),index(j))
                end do
                index(l+1)=index(j)
                index(j)=indext
                jstack=jstack+2
                if (jstack > NSTACK) call nrerror('indexx: NSTACK too small')
                if (r-i+1 >= j-l) then
                        istack(jstack)=r
                        istack(jstack-1)=i
                        r=j-1
```

[indexx.f90]

```
                    else
                        istack(jstack)=j-1
                        istack(jstack-1)=l
                        l=i
                    end if
                end if
            end do
            CONTAINS
!BL
            SUBROUTINE icomp_xchg(i,j)
            INTEGER(I4B), INTENT(INOUT) :: i,j
            INTEGER(I4B) :: swp
            if (iarr(j) < iarr(i)) then
                swp=i
                i=j
                j=swp
            end if
            END SUBROUTINE icomp_xchg
            END SUBROUTINE indexx_i4b
```

```
!*****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!*****************************************
! Time-stamp: <98/04/04 15:26:58 ayahil>

!!! IDL-like functions INDGEN & FINDGEN returning the integers from 1 to n.
!!! Note the important difference: the series begin with 1, not 0, to conform
!!! to standard FORTRAN usage.

FUNCTION Indgen( n ) RESULT( out )

  IMPLICIT NONE
                    ! Arguments
  INTEGER, INTENT(in) :: n
  INTEGER, DIMENSION(n) :: out
                    ! Locals
  INTEGER :: i

  out = (/ (i, i=1,n) /)

END FUNCTION Indgen

FUNCTION Findgen( n ) RESULT( out )

  USE Nrtype
  IMPLICIT NONE
                    ! Arguments
  INTEGER, INTENT(in) :: n
  REAL(sp), DIMENSION(n) :: out
                    ! Locals
  INTEGER :: i

  out = (/ (i, i=1,n) /)

END FUNCTION Findgen
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/26 21:43:38 ayahil>

MODULE Interfaces

 INTERFACE
  SUBROUTINE Algebron( abstol, add_iter, chi2, gof, gof_ev, gof_std, n_t, &
    & nsig, p_t, prn, prob, r_t, reltol, signif, stepmx, v, w_t, z )
   USE Nrtype
   INTEGER, INTENT(in) :: add_iter, prn, stepmx
   INTEGER, INTENT(in), TARGET :: n_t, p_t
   REAL(sp), INTENT(in) :: abstol, nsig, reltol, signif
   REAL(sp), INTENT(in), TARGET, DIMENSION(n_t) :: w_t
   REAL(sp), INTENT(in), TARGET, DIMENSION(n_t,p_t) :: r_t
   REAL(sp), INTENT(inout), DIMENSION(p_t*(p_t+1)) :: z
   REAL(sp), INTENT(out) :: gof, gof_ev, gof_std
   REAL(sp), INTENT(out), DIMENSION(n_t) :: chi2, prob
   REAL(sp), INTENT(out), DIMENSION(p_t,p_t) :: v
  END SUBROUTINE Algebron
 END INTERFACE

 INTERFACE
  FUNCTION Cml( lambda, sig, dlambda ) RESULT( out )
   USE Nrtype
   USE Parm, ONLY: k, p
   REAL(sp), INTENT(in), DIMENSION(:) :: sig
   REAL(sp), INTENT(in), DIMENSION(:,:) :: lambda
   REAL(sp), OPTIONAL, INTENT(out), DIMENSION(p,k) :: dlambda
   REAL(sp) :: out
  END FUNCTION Cml
 END INTERFACE

 INTERFACE
  FUNCTION Covar( z ) RESULT( out )
   USE Nrtype
   USE Parm, ONLY: p
   REAL(sp), INTENT(in), DIMENSION(:) :: z
   REAL(sp), DIMENSION(p,p) :: out
  END FUNCTION Covar
 END INTERFACE

 INTERFACE
```

```
  FUNCTION Cull( ln, nsig, z ) RESULT( out )
   USE Parm, ONLY: k, p
   USE Nrtype
   REAL(sp), INTENT(in) :: nsig
   REAL(sp), INTENT(in), DIMENSION(:) :: z
   REAL(sp), INTENT(in), DIMENSION(:) :: ln
   REAL(sp), DIMENSION(k*p) :: out
  END FUNCTION Cull
END INTERFACE

INTERFACE
  FUNCTION Func( z, xi ) RESULT( out )
   USE Nrtype
   REAL(sp), INTENT(in), TARGET, DIMENSION(:) :: z
   REAL(sp), OPTIONAL, INTENT(inout), TARGET, DIMENSION(SIZE(z)) :: xi
   REAL(sp) :: out
  END FUNCTION Func
END INTERFACE

INTERFACE
  SUBROUTINE Gen_covar( c, p )
   USE Nrtype
   INTEGER, INTENT(in) :: p
   REAL(sp), INTENT(out), DIMENSION(p,p) :: c
  END SUBROUTINE Gen_covar
END INTERFACE

INTERFACE
  SUBROUTINE Gen_dev( c, n, p, x )
   USE Nrtype
   INTEGER, INTENT(in) :: n, p
   REAL(sp), INTENT(in), DIMENSION(p,p) :: c
   REAL(sp), INTENT(out), DIMENSION(n,p) :: x
  END SUBROUTINE Gen_dev
END INTERFACE

INTERFACE
  FUNCTION Get_gof( chi2, gof_ev, gof_std, prob, r, v, w ) RESULT( out )
   USE Nrtype
   USE Parm, ONLY: n
   REAL(sp), INTENT(in), DIMENSION(:) :: w
   REAL(sp), INTENT(in), DIMENSION(:,:) :: r, v
   REAL(sp), INTENT(out) :: gof_ev, gof_std
   REAL(sp), INTENT(out), DIMENSION(n) :: chi2, prob
   REAL(sp) :: out
  END FUNCTION Get_gof
```

```
END INTERFACE

INTERFACE
  FUNCTION Ml( lambda, sig, dlambda, dsig ) RESULT( out )
   USE Nrtype
   USE Parm, ONLY: k, p
   REAL(sp), INTENT(in), DIMENSION(:) :: sig
   REAL(sp), INTENT(in), DIMENSION(:,:) :: lambda
   REAL(sp), OPTIONAL, INTENT(out), DIMENSION(p) :: dsig
   REAL(sp), OPTIONAL, INTENT(out), DIMENSION(p,k) :: dlambda
   REAL(sp) :: out
  END FUNCTION Ml
END INTERFACE

INTERFACE
  FUNCTION Next( z ) RESULT( out )
   USE Parm, ONLY: p
   USE Nrtype
   REAL(sp), INTENT(in), DIMENSION(:) :: z
   REAL(sp), DIMENSION(p) :: out
  END FUNCTION Next
END INTERFACE

INTERFACE
  SUBROUTINE Prepare( cntmn, idx, n, p, p0, prune, r, ravg, w, x )
   USE Nrtype
   INTEGER, INTENT(in) :: cntmn, n, p0
   INTEGER, INTENT(out) :: p
   INTEGER, INTENT(out), DIMENSION(p0) :: idx
   REAL(sp), INTENT(in) :: prune
   REAL(sp), INTENT(in), DIMENSION(n) :: w
   REAL(sp), INTENT(in), DIMENSION(n,p0) :: x
   REAL(sp), INTENT(out), DIMENSION(n,p0) :: r
   REAL(sp), INTENT(out), DIMENSION(p0) :: ravg
  END SUBROUTINE Prepare
END INTERFACE

INTERFACE
  SUBROUTINE Sample( ln, r, s, w )
   USE Nrtype
   USE Parm, ONLY: p
   REAL(sp), INTENT(in), DIMENSION(:) :: w
   REAL(sp), INTENT(in), DIMENSION(:,:) :: r
   REAL(sp), INTENT(out), DIMENSION(p) :: ln, s
  END SUBROUTINE Sample
END INTERFACE
```

```
END MODULE Interfaces
```

[interfaces.f90]

```
                    !*******************************************
                    ! Copyright (C) 1997 by Amos Yahil.
                    ! All Rights Reserved.
                    ! Based on (C) Numerical Recipes software.
                    !*******************************************
                    ! Time-stamp: <98/06/07 14:46:19 ayahil>
MODULE f1dim_mod
 USE nrtype
 REAL(SP), DIMENSION(:), POINTER :: pcom,xicom
CONTAINS
                    !BL
 FUNCTION f1dim(x,df)
  IMPLICIT NONE
  REAL(SP), INTENT(IN) :: x
  REAL(SP), OPTIONAL, INTENT(OUT) :: df
  REAL(SP) :: f1dim
  INTERFACE
    FUNCTION func(x,g)
     USE nrtype
     REAL(SP), DIMENSION(:), INTENT(IN) :: x
     REAL(SP), DIMENSION(SIZE(x)), OPTIONAL, INTENT(OUT) :: g
     REAL(SP) :: func
    END FUNCTION func
  END INTERFACE
  REAL(SP), DIMENSION(SIZE(pcom)) :: g
  IF(PRESENT(df)) THEN
    f1dim=func(pcom+x*xicom,g)
    df=DOT_PRODUCT(g,xicom)
  ELSE
    f1dim=func(pcom+x*xicom)
  END IF
 END FUNCTION f1dim
END MODULE f1dim_mod

SUBROUTINE linmin(p,xi,fret,dx,lop,hip)
 USE nrtype; USE nrutil, ONLY : assert_eq,imaxloc,iminloc,nrerror
 USE mynr, ONLY : mnbrak,brent
 USE f1dim_mod
 IMPLICIT NONE
 REAL(SP), INTENT(OUT) :: fret
 REAL(SP), DIMENSION(:), TARGET, INTENT(INOUT) :: p,xi
 REAL(SP), OPTIONAL :: dx
 REAL(SP), DIMENSION(:), OPTIONAL, INTENT(in) :: hip,lop
 INTEGER(I4B) :: in,ix,np
 REAL(SP) :: ax,bx,df,fa,fb,fx,xmin,xx
 REAL(SP), DIMENSION(SIZE(p)) :: hix,lox
```

```
                        !Check dimensions
  np=assert_eq(SIZE(p),SIZE(xi),'linmin')
                        !Set limits
  lox = -HUGE(1.0_sp)
  hix = HUGE(1.0_sp)
  IF (PRESENT(lop)) THEN
    np=assert_eq(np,SIZE(lop),'linmin: lop size')
    IF (MINVAL(p-lop) < 0.0) THEN
      WRITE(0,*) p
      CALL nrerror('linmin: p < lop')
    END IF
    WHERE (lop > -HUGE(1.0_sp) .AND. xi < 0.0)
      hix=(lop-p)/xi
    END WHERE
    WHERE (lop > -HUGE(1.0_sp) .AND. xi > 0.0)
      lox=(lop-p)/xi
    END WHERE
  END IF
  IF (PRESENT(hip)) THEN
    np=assert_eq(np,SIZE(hip),'linmin: hip size')
    IF (MAXVAL(p-hip) > 0.0) THEN
      WRITE(0,*) p
      CALL nrerror('linmin: p > hip')
    END IF
    WHERE (hip < HUGE(1.0_sp) .AND. xi > 0.0)
      hix=(hip-p)/xi
    END WHERE
    WHERE (hip > HUGE(1.0_sp) .AND. xi < 0.0)
      lox=(hip-p)/xi
    END WHERE
  END IF
                        !Variables passed to f1dim
  pcom => p
  xicom => xi
                        !Prepare limits for mnbrak
  in=iminloc(hix)
  ix=imaxloc(lox)
  ax=0.0
  IF (PRESENT(dx)) THEN
    xx=MIN(dx,0.5_sp*hix(in))
  ELSE
    xx=MIN(1.0_sp,0.5*hix(in))
  END IF
  fa=F1dim(ax)
  DO WHILE (F1dim( xx ) > fa)
    xx=0.5*xx
```

```
  END DO
                    !Bracket minimization limits
  CALL mnbrak(ax,xx,bx,fa,fx,fb,f1dim,lox=lox(ix),hix=hix(in))
                    !Check for minimum at boundaries.  The
                    ! parameter which forces the boundary should
                    ! be set exactly to avoid roundoff error.
  IF (bx == lox(ix)) THEN
    fret=f1dim(bx,df)
    IF (df > 0.0) THEN
      xi=bx*xi
      p=p+xi
      IF (xi(ix) < 0.0) THEN
        p(ix)=lop(ix)
      ELSE
        p(ix)=hip(ix)
      END IF
      RETURN
    END IF
  END IF
  IF (bx == hix(in)) THEN
    fret=f1dim(bx,df)
    IF (df < 0.0) THEN
      xi=bx*xi
      p=p+xi
      IF (xi(in) < 0.0) THEN
        p(in)=lop(in)
      ELSE
        p(in)=hip(in)
      END IF
      RETURN
    END IF
  END IF
                    !Minimize within interval
  fret=brent(ax,xx,bx,f1dim,SQRT(EPSILON(1.0_sp)),xmin)
                    !Update the parameters
  xi=xmin*xi
  p=p+xi
END SUBROUTINE linmin
```

```
/*****************************************************************/
/*                                                               */
/*     Copyright (C) 1998 Rainbow Technologies, Inc.             */
/*                  All Rights Reserved                          */
/*                                                               */
/*    This Module contains Proprietary Information of Rainbow    */
/*  Technologies, Inc., and should be treated as Confidential.   */
/*****************************************************************/

#ifndef _LSERV_H_
#define _LSERV_H_

/*H*****************************************************************
* FILENAME    : lserv.h
*
* DESCRIPTION :
*       Contains public function prototypes, macros and defines
*       needed for licensing an app using SentinelLM library.
*
* USAGE       :
*       This file should be included by all users of SentinelLM
*       client library.
* NOTES       :
*
*H*/

#ifdef __cplusplus
 extern "C" {
#endif

#include <stdio.h>     /* For definition of FILE*  */

/*
 * Developer should compile with _VWIN31X_ for 16 bit Windows (Windows 3.1x).
 * Developer can compile with LSNOPROTO to force no prototyping.
 */

#if defined (_VWIN31X_)
  /* MS Windows 3.1x application */
# define VMSWINAPI _far _pascal
# define LSFAR     _far
# define VDLL32
#elif defined(_VWIN95_) || defined(_VWINNT_)
  /* 32-bit MS Windows application */
  /* This section is for internal use.  Do not define _VWIN95_ or _VWINNT_ */
# define VMSWINAPI
```

```
# define LSFAR
# define VDLL32    __declspec(dllexport)
#else
# define VMSWINAPI
# define LSFAR
# define VDLL32
#endif /* MS Windows */


/*-------------------------------------------------------------------------*/
/* To inactivate licensing completely, use the following macro which       */
/* will make all SentinelLM functions void:                                */
/*-------------------------------------------------------------------------*/

/*
#define NO_LICENSE
*/

/*-------------------------------------------------------------------------*/
/* LSAPI constants                                                         */
/*-------------------------------------------------------------------------*/

#define  LS_DEFAULT_UNITS        (unsigned long)0xFFFFFFFF
#define  LS_ANY                  ((unsigned char LSFAR *)NULL)
#define  LS_USE_LAST             (unsigned long)0x0800FFFF

/*-------------------------------------------------------------------------*/
/* Standalone mode constants                                               */
/*-------------------------------------------------------------------------*/

#define VLS_STANDALONE           "no-net"

/*-------------------------------------------------------------------------*/
/* Trace level                                                             */
/*-------------------------------------------------------------------------*/

#define VLS_NO_TRACE             1  /* This is the default value */
#define VLS_TRACE_KEYS           2
#define VLS_TRACE_FUNCTIONS      4
#define VLS_TRACE_ERRORS         8
#define VLS_TRACE_ALL            16

/*-------------------------------------------------------------------------*/
/* Error handling                                                          */
/*-------------------------------------------------------------------------*/
```

```
#define VLS_ON              1 /* This is the default value */
#define VLS_OFF             0

/*-------------------------------------------------------------------*/
/* Error handling                                                    */
/*-------------------------------------------------------------------*/
#define VLS_EH_SET_ALL      0

/*-------------------------------------------------------------------*/
/* True/False                                                        */
/*-------------------------------------------------------------------*/

#define VLS_TRUE            0
#define VLS_FALSE           1

/*-------------------------------------------------------------------*/
/* Sharing criteria                                                  */
/*-------------------------------------------------------------------*/

#define VLS_NO_SHARING              0
#define VLS_USER_NAME_ID            1
#define VLS_CLIENT_HOST_NAME_ID     2
#define VLS_X_DISPLAY_NAME_ID       3
#define VLS_VENDOR_SHARED_ID        4
#define VLS_NO_SHARING_STRING       "0"
#define VLS_USER_NAME_ID_STRING     "1"
#define VLS_CLIENT_HOST_NAME_ID_STRING  "2"
#define VLS_X_DISPLAY_NAME_ID_STRING    "3"
#define VLS_VENDOR_SHARED_ID_STRING     "4"

/*-------------------------------------------------------------------*/
/* Holding criteria                                                  */
/*-------------------------------------------------------------------*/

#define VLS_HOLD_NONE       0
#define VLS_HOLD_VENDOR     1
#define VLS_HOLD_CODE       2
#define VLS_HOLD_NONE_STRING    "0"
#define VLS_HOLD_VENDOR_STRING  "1"
#define VLS_HOLD_CODE_STRING    "2"

/*-------------------------------------------------------------------*/
/* Client-server lock mode                                           */
/*-------------------------------------------------------------------*/

#define VLS_NODE_LOCKED     0
```

```
#define VLS_FLOATING            1
#define VLS_DEMO_MODE           2
#define VLS_CLIENT_NODE_LOCKED  3

/*-----------------------------------------------------------------------*/
/* Locking criteria                                                      */
/*-----------------------------------------------------------------------*/

/* Test whether a particular locking criterion is being used. */
#define VLS_LOCK_TO_ID_PROM(V)       (( (V) >> 0 ) & 0x1)
#define VLS_LOCK_TO_IP_ADDR(V)       (( (V) >> 1 ) & 0x1)
#define VLS_LOCK_TO_DISK_ID(V)       (( (V) >> 2 ) & 0x1)
#define VLS_LOCK_TO_HOSTNAME(V)      (( (V) >> 3 ) & 0x1)
#define VLS_LOCK_TO_ETHERNET(V)      (( (V) >> 4 ) & 0x1)
#define VLS_LOCK_TO_NW_IPX(V)        (( (V) >> 5 ) & 0x1)
#define VLS_LOCK_TO_NW_SERIAL(V)     (( (V) >> 6 ) & 0x1)
#define VLS_LOCK_TO_PORTABLE_SERV(V) (( (V) >> 7 ) & 0x1)
#define VLS_LOCK_TO_CUSTOM(V)        (( (V) >> 8 ) & 0x1)

/* To set a particular locking criterion. */
#define VLS_LOCK_ID_PROM        0x1
#define VLS_LOCK_IP_ADDR        0x2
#define VLS_LOCK_DISK_ID        0x4
#define VLS_LOCK_HOSTNAME       0x8
#define VLS_LOCK_ETHERNET       0x10
#define VLS_LOCK_NW_IPX         0x20
#define VLS_LOCK_NW_SERIAL      0x40
#define VLS_LOCK_PORTABLE_SERV  0x80
#define VLS_LOCK_CUSTOM         0x100
/* Highest bit currently in use : */
#define VLS_LOCK_HIGHEST_BIT    9    /* Starting from 1... */
/* Mask with all locking criteria set. */
#define VLS_LOCK_ALL            0x1FF


/*-----------------------------------------------------------------------*/
/* License does not have an expiration date                              */
/*-----------------------------------------------------------------------*/

#define VLS_NO_EXPIRATION       -1

/*-----------------------------------------------------------------------*/
/* This number represents infinite keys                                  */
/*-----------------------------------------------------------------------*/

#define VLS_INFINITE_KEYS       0xffff
```

```
#define VLS_INFINITE_KEYS_STRING  ""


/*-------------------------------------------------------------------*/
/* Type definitions                                                  */
/*-------------------------------------------------------------------*/

typedef unsigned long          LS_STATUS_CODE;
typedef unsigned long          LS_HANDLE;

#define MAX_NAME_LEN 128
#define MAX_BUF_LEN  512

#define VLS_DISC_NO_OPTIONS     0
#define VLS_DISC_RET_ON_FIRST   1
#define VLS_DISC_PRIORITIZED_LIST 2
#define VLS_DISC_NO_USERLIST    4
#define VLS_DISC_DEFAULT_OPTIONS VLS_DISC_NO_OPTIONS

#define NO_RET_ON_FIRST 0
#define RET_ON_FIRST 1

typedef enum {VLS_LOCAL_UPD_ENABLE, VLS_LOCAL_UPD_DISABLE}
VLS_LOC_UPD_STAT;


/*-------------------------------------------------------------------*/
/* Challenge, ChallengeResponse structs                              */
/*-------------------------------------------------------------------*/

typedef struct {
  unsigned long  ulReserved;
  unsigned long  ulChallengedSecret;
  unsigned long  ulChallengeSize;
  unsigned char  ChallengeData[30];
} CHALLENGE, LS_CHALLENGE;

typedef struct {
  unsigned long ulResponseSize;
  unsigned char ResponseData[16];
} CHALLENGERESPONSE;

/*-------------------------------------------------------------------*/
/* Client and feature information structures                         */
/* To be used in VLSgetClientInfo, VLSgetFeatureInfo and VLShandleInfo   */
/*-------------------------------------------------------------------*/
```

```
#define MAXFEALEN     64
#define MAXLEN        MAXFEALEN
#define SITEINFOLEN   150
#define VENINFOLEN    100
#define MAXCLLOCKLEN  200

/* Client Information structure */
struct client_info_struct {
  char        user_name[MAXLEN];
  unsigned long host_id;
  char        group[MAXLEN];
  long        start_time;
  long        hold_time;
  long        end_time;
  long        key_id;
  char        host_name[MAXLEN];
  char        x_display_name[MAXLEN];
  char        shared_id_name[MAXLEN];
  int         num_units;
  int         q_wait_time;
  int         is_holding;          /* VLS_TRUE/VLS_FALSE        */
  int         is_sharing;          /* # of clients using this key */
};
typedef  struct client_info_struct       VLSclientInfo;

/* Feature Information structure */
struct feature_info_struct {
  char    feature_name[MAXFEALEN];
  char    version[MAXFEALEN];
  long    birth_day;
  long    death_day;
  int     num_licenses;
  int     total_resv;
  int     lic_from_resv;
  int     lic_from_free_pool;
  int     is_node_locked;       /* VLS_FLOATING/VLS_NODE_LOCKED/... */
  int     concurrency;
  int     sharing_crit;
  int     locking_crit;
  int     holding_crit;
  int     num_subnets;
  char    site_license_info[SITEINFOLEN];
  long    hold_time;
  int     meter_value;
  char    vendor_info[VENINFOLEN];
  char    cl_lock_info[MAXCLLOCKLEN];
```

```
 long   key_life_time;
 int    sharing_limit;
 int    soft_num_licenses;
 int    is_standalone;        /* VLS_TRUE/VLS_FALSE */
 int    check_time_tamper;    /* VLS_TRUE/VLS_FALSE */
 int    is_additive;          /* VLS_TRUE/VLS_FALSE */
};
typedef  struct feature_info_struct       VLSfeatureInfo;

/*-------------------------------------------------------------------------*/
/* Client version information structure                                    */
/* To be used in VLSgetLibInfo                                             */
/*-------------------------------------------------------------------------*/

/* VLSgetLibInfo() should return the same version string in szVersion: */
#define LS_VERSION   "6.0"

#define LS_PROD_NAME "SentinelLM"

#define LS_COPYRIGHT \
"  Copyright (C) 1998 Rainbow Technologies, Inc.\n\n"

#define LIBINFOLEN  32

typedef struct {
 unsigned long ulInfoCode;
 char       szVersion  [LIBINFOLEN];
 char       szProtocol [LIBINFOLEN];
 char       szPlatform [LIBINFOLEN];
 char       szUnused1  [LIBINFOLEN];
 char       szUnused2  [LIBINFOLEN];
} LS_LIBVERSION;

/*-------------------------------------------------------------------------*/
/* install info structure holds setup information for Sentinel LM          */
/* To be used in VLSinstall                                                */
/*-------------------------------------------------------------------------*/

typedef struct {
   long size;
   char * regKey;
   char * regValue;
   char * installDir;
   long time1;
   long time2;
   long reserved1;
```

```
} VLSinstallInfo;

/*-------------------------------------------------------------------------*/
/* Macros for status codes                                                 */
/* prefix LS  :  LSAPI status codes                                        */
/* prefix VLS :  Our own status codes                                      */
/*-------------------------------------------------------------------------*/

/* The function completed successfully                                     */
#define LS_SUCCESS              0x0

/* Handle used on call did not describe a valid licensing system context  */
#define LS_BADHANDLE            (LS_STATUS_CODE)0xC8001001

/* Licensing system could not locate enough available licensing resources */
/* to satisfy the request                                                  */
#define LS_INSUFFICIENTUNITS    (LS_STATUS_CODE)0xC8001002

/* No licensing system could be found with which to perform the function  */
/* invoked                                                                 */
#define LS_LICENSESYSNOTAVAILABLE  (LS_STATUS_CODE)0xC8001003

/* The licensing system has determined that the resources used to satisfy */
/* a previous request are no longer granted to the calling software       */
#define LS_LICENSETERMINATED    (LS_STATUS_CODE)0xC8001004

/* The licensing system has no licensing resources that could satisfy the */
/* request.                                                                */
#define LS_NOAUTHORIZATIONAVAILABLE (LS_STATUS_CODE)0xC8001005

/* The licensing system has licensing resources that could satisfy the    */
/* request, but they are not available at the time of the request         */
#define LS_NOLICENSESAVAILABLE  (LS_STATUS_CODE)0xC8001006

/* Insufficient resources (such as memory) are available to complete the  */
/* request                                                                 */
#define LS_NORESOURCES          (LS_STATUS_CODE)0xC8001007

/* The network is unavailable                                              */
#define LS_NO_NETWORK           (LS_STATUS_CODE)0xC8001008

/* A warning occured while looking up an error messge string for the      */
/* LSGetMessage() function                                                 */
#define LS_NO_MSG_TEXT          (LS_STATUS_CODE)0xC8001009

/* An unrecognized status code was passed into the LSGetMessage() function*/
```

```
#define LS_UNKNOWN_STATUS          (LS_STATUS_CODE)0xC800100A

/* An invalid index was specified in LSEnumProviders() or LSQuery License */
#define LS_BAD_INDEX               (LS_STATUS_CODE)0xC800100B

/* No additional units are available                                      */
#define LS_NO_MORE_UNITS           (LS_STATUS_CODE)0xC800100C

/* The license associated with the current context has expired. This may  */
/* be due to a time-restriction on the license                            */
#define LS_LICENSE_EXPIRED         (LS_STATUS_CODE)0xC800100D

/* Input buffer is too small, need a bigger buffer                        */
#define LS_BUFFER_TOO_SMALL        (LS_STATUS_CODE)0xC800100E

/* No success in achieving the target                                     */
#define LS_NO_SUCCESS              (LS_STATUS_CODE)0xC800100F


/* 1. Generic error when a license is denied by a server.
 * If reasons are known, more specific errors are given */
#define VLS_NO_LICENSE_GIVEN       1

/* 2. Application has not been given a name. */
#define VLS_APP_UNNAMED            2

/* 3. Unkown host (Application is given a server name but that
 * server name doesnt seem to exist) */
#define VLS_HOST_UNKNOWN           3

/* 4. No FILE giving license server name (Application cannot figure
 * out the license server. */
#define VLS_NO_SERVER_FILE         4

/* 5. On the specified machine, license server is not RUNNING. */
#define VLS_NO_SERVER_RUNNING      5

/* 6. This /feature is node locked but the request for a key came
 * from a machine other than the host running the SentinelLM server. */
#define VLS_APP_NODE_LOCKED        6

/* 7. LSrelease called when this copy of the application had not
 * received a valid key from the SentinelLM server. */
#define VLS_NO_KEY_TO_RETURN       7

/* 8. Failed to return the key issued to this copy of the application */
```

```
#define VLS_RETURN_FAILED          8

/* 9. End of clients on calling VLSgetClientInfo */
#define VLS_NO_MORE_CLIENTS        9

/* 10. End of features on calling VLSgetFeatureInfo */
#define VLS_NO_MORE_FEATURES       10

/* 11. General error by vendor in calling function etc. */
#define VLS_CALLING_ERROR          11

/* 12. Internal error in SentinelLM */
#define VLS_INTERNAL_ERROR         12

/* 13. Irrecoverable Internal error in SentinelLM */
#define VLS_SEVERE_INTERNAL_ERROR  13

/* 14. On the specified machine, license server is not responding.
 * (Probable cause - network down, wrong port number, some other
 * application on that port etc.) */
#define VLS_NO_SERVER_RESPONSE     14

/* 15. User/machine excluded */
#define VLS_USER_EXCLUDED          15

/* 16. Unknown shared id */
#define VLS_UNKNOWN_SHARED_ID      16

/* 17. No servers responded to client broadcast */
#define VLS_NO_RESPONSE_TO_BROADCAST 17

/* 18. No such feature recognized by server */
#define VLS_NO_SUCH_FEATURE        18

/* 19. Failed to add license */
#define VLS_ADD_LIC_FAILED         19

/* 20. Failed to delete license */
#define VLS_DELETE_LIC_FAILED      20

/* 21. Last update was done locally */
#define VLS_LOCAL_UPDATE           21

/* 22. Last update was done by the SentinelLM server */
#define VLS_REMOTE_UPDATE          22
```

```
/* 23. The vendor identification of requesting application does not
 * match with that of the application licensed by this system. */
#define VLS_VENDORIDMISMATCH        23

/* 24. The server has licenses for the same feature,version from multiple
 * vendors, and it is not clear from the requested operation which license
 * the requestor is interested in. */
#define VLS_MULTIPLE_VENDORID_FOUND 24

/* 25. An error has occured in decrypting (or decoding) a network message.
 * Probably an incompatible or unknown server, or a version mismatch. */
#define VLS_BAD_SERVER_MESSAGE      25

/* 26. The server has found evidence of tampering of the system clock,
 * and it cannot service the request since the license for this feature
 * has been set to be time tamper proof. */
#define VLS_CLK_TAMP_FOUND          26

/* 27. The specified operation is not permitted - authorization failed. */
#define VLS_NOT_AUTHORIZED          27

/* 28. The domain of server is different from that of client. */
#define VLS_INVALID_DOMAIN          28


/*-------------------------------------------------------------------------*/
/* Function Prototypes                                                     */
/*-------------------------------------------------------------------------*/


VDLL32 LS_STATUS_CODE VMSWINAPI LSRequest (
#ifndef LSNOPROTO
  unsigned char  LSFAR *license_system,
  unsigned char  LSFAR *publisher_name,
  unsigned char  LSFAR *product_name,
  unsigned char  LSFAR *version,
  unsigned long  LSFAR *units_reqd,
  unsigned char  LSFAR *log_comment,
  LS_CHALLENGE   LSFAR *challenge,
  LS_HANDLE      LSFAR *lshandle
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI LSRelease (
#ifndef LSNOPROTO
  LS_HANDLE          lshandle,
```

```
 unsigned long       unused1,
 unsigned char  LSFAR *log_comment
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI LSUpdate (
#ifndef LSNOPROTO
 LS_HANDLE          lshandle,
 unsigned long       unused1,
 long        LSFAR *unused2,
 unsigned char  LSFAR *unused3,
 LS_CHALLENGE   LSFAR *unused4
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSbatchUpdate (
#ifndef LSNOPROTO
 int              numHandles,    /* IN                    */
 LS_HANDLE     LSFAR * lshandle,      /* INOUT - numHandles elements */
 unsigned long  LSFAR * unused1,       /* IN    - should be NULL      */
 long         LSFAR * unused2,       /* IN    - should be NULL      */
 unsigned char  LSFAR * unused3,       /* IN    - should be NULL      */
 LS_CHALLENGE   LSFAR * unused4,       /* IN    - should be NULL      */
 LS_STATUS_CODE LSFAR * status         /* OUT   - numHandles elements */
#endif /* LSNOPROTO */
);


VDLL32 LS_STATUS_CODE VMSWINAPI VLSshutDown (
#ifndef LSNOPROTO
 char   LSFAR   *hostName
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI LSGetMessage (
#ifndef LSNOPROTO
 LS_HANDLE          lshandle,
 LS_STATUS_CODE       Value,
 unsigned char  LSFAR *Buffer,
 unsigned long       BufferSize
#endif
);

/* Single-call licensing. */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSlicense(
```

```
#ifndef LSNOPROTO
 unsigned char  LSFAR *feature_name,
 unsigned char  LSFAR *version,
 LS_HANDLE     LSFAR *handle
#endif  /* LSNOPROTO */
);

/* Disables single-call licensing; returns license key. */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSdisableLicense(
#ifndef LSNOPROTO
 LS_HANDLE     LSFAR *handle
#endif  /* LSNOPROTO */
);

/*-------------------------------------------------------------------------*/
/* Disables automatic renewal of license                                   */
/* call with handle to disable automatic renewal of one feature            */
/* call with (LS_HANDLE) 0 to disable auto renewal of all features         */
/* on UNIX, call VLSdisableAutoTimer before using sleep                    */
/* on Win32, call VLSdisableAutoTimer when thread has no message loop      */
/*-------------------------------------------------------------------------*/
VDLL32 LS_STATUS_CODE VMSWINAPI VLSdisableAutoTimer(
#ifndef LSNOPROTO
 LS_HANDLE handle,
 int      state       /* VLS_ON or VLS_OFF */
#endif  /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetTraceLevel(
#ifndef LSNOPROTO
 int trace_level
#endif /* LSNOPROTO*/
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetContactServer(
#ifndef LSNOPROTO
 char LSFAR *server_name
#endif /* LSNOPROTO */
);

/* THIS FUNCTION IS OBSOLETE.  Use VLSsetContactServer() instead. */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetServerName(
#ifndef LSNOPROTO
 char LSFAR *server_name
#endif /* LSNOPROTO */
);
```

```
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetServerList (
#ifndef LSNOPROTO
 char LSFAR *outBuf,
 int      outBufSz
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSinitServerList(
#ifndef LSNOPROTO
 char LSFAR *ServerList,
 int      option_flag
#endif /* LSNOPROTO */
);

/* Get the name of license server. */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetContactServer(
#ifndef LSNOPROTO
 char  LSFAR   *outBuf,
 int   outBufSz
#endif
);

/* Get the name of license server. */
/* THIS FUNCTION IS OBSOLETE.  Use VLSgetContactServer() instead. */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetServerName (
#ifndef LSNOPROTO
 char  LSFAR   *outBuf,
 int   outBufSz
#endif
);

/* Get the name of license server from Handle. */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetServerNameFromHandle (
#ifndef LSNOPROTO
 LS_HANDLE     handle_id,
 char  LSFAR   *outBuf,
 int   outBufSz
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSerrorHandle (
#ifndef LSNOPROTO
 int errorHandle
#endif
);
```

```
/*
 * Replaces the default error handler for the specified error.
 * Error Handlers are automatically called on error, unless disabled.
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetErrorHandler(
#ifndef LSNOPROTO
  LS_STATUS_CODE (VMSWINAPI * myErrorHandler)(LS_STATUS_CODE, char LSFAR
*),
  LS_STATUS_CODE LS_ErrorType
#endif /* LSNOPROTO */
);

/*
 * Configures displaying of error msgs to the user through the default
 * error handlers.  If you disable the default error handlers you do not
 * need to use this function.
 * Default behavior:
 *   Windows - Pop up a Message Box.
 *   Unix    - Write to stderr.
 * You can alter this behavior by providing either a FILE* or a file path.
 * The other parameter should be NULL.
 * If you provide both, preference will be given to the FILE*.
 */
typedef enum {
  VLS_STDOUT, VLS_STDERR
} VLS_ERR_FILE;

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetUserErrorFile(
#ifndef LSNOPROTO
  VLS_ERR_FILE msgFile,  /* IN - Desired error file */
  char LSFAR * filePath  /* IN - Full path of desired error file */
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetRemoteRenewalTime(
#ifndef LSNOPROTO
  unsigned char LSFAR *feature_name,
  unsigned char LSFAR *version,
  int renewal_time     /* renewal time in secs */
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSdiscover (
#ifndef LSNOPROTO
  unsigned char LSFAR *feature_name,
  unsigned char LSFAR *version,
```

```
 unsigned char LSFAR *unused1,
 int             bufferSize,
 char       LSFAR *server_names,
 int             broadcastFlag,
 char       LSFAR *vendor_list
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSwhere (
#ifndef LSNOPROTO
 unsigned char LSFAR *feature_name,
 unsigned char LSFAR *version,
 unsigned char LSFAR *unused1,
 int             bufferSize,
 char       LSFAR *server_names,
 int             broadcastFlag
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSaddFeature (
#ifndef LSNOPROTO
 unsigned char LSFAR *license_string,
 unsigned char LSFAR *unused1,
 LS_CHALLENGE  LSFAR *unused2
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSaddFeatureToFile (
#ifndef LSNOPROTO
 unsigned char LSFAR *license_string,
 unsigned char LSFAR *unused1,
 unsigned char LSFAR *unused2,
 LS_CHALLENGE  LSFAR *unused3
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSdeleteFeature (
#ifndef LSNOPROTO
 unsigned char LSFAR *feature_name,
 unsigned char LSFAR *version,
 unsigned char LSFAR *unused1,
 LS_CHALLENGE  LSFAR *unused2
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetVersions (
```

```
#ifndef LSNOPROTO
  char        LSFAR *feature_name,
  int             bufferSize,
  char        LSFAR *versionList,
  char        LSFAR *unused1
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetHandleInfo (
#ifndef LSNOPROTO
  LS_HANDLE          lshandle,
  VLSclientInfo LSFAR *client_info
#endif
);

/* Get information about client */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetClientInfo (
#ifndef LSNOPROTO
  unsigned char   LSFAR *feature_name,
  unsigned char   LSFAR *version,
  int             index,
  char          LSFAR *unused1,
  VLSclientInfo   LSFAR *client_info
#endif /* LSNOPROTO */
);

/* Get information about feature */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetFeatureInfo(
#ifndef LSNOPROTO
  unsigned char   LSFAR *feature_name,
  unsigned char   LSFAR *version,
  int             index,
  char          LSFAR *unused1,
  VLSfeatureInfo  LSFAR *feature_info
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetFeatureFromHandle (
#ifndef LSNOPROTO
  LS_HANDLE          lshandle,
  char          LSFAR *Buffer,
  unsigned long       BufferSize
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetVersionFromHandle (
```

```
#ifndef LSNOPROTO
 LS_HANDLE          lshandle,
 char        LSFAR *Buffer,
 unsigned long      BufferSize
#endif
);

/*
 * Note that the information returned by this function will be correct
 * only immediately after acquiring the handle.  The information in the
 * handle is NOT updated subsequently.
 *
 * The function is used when the clocks may not be in sync. It
 * returns the difference in seconds between the estimated current
 * time on the server and the estimated time on the client.
 * The estimation error is usually the network latency time.
 *
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetTimeDriftFromHandle (
#ifndef LSNOPROTO
 LS_HANDLE          lshandle,              /* IN  */
 long       LSFAR *secondsServerAheadOfClient   /* OUT */
#endif
);


/*
 * Note that the information returned by this function will be correct
 * only immediately after acquiring the handle.  The information in the
 * handle is NOT updated subsequently.
 *
 * The function is used when the clocks may not be in sync. It
 * returns the difference in seconds between the estimated current
 * time on the server and the estimated feature expiration time
 * on the server.
 *
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetFeatureTimeLeftFromHandle (
#ifndef LSNOPROTO
 LS_HANDLE          lshandle,              /* IN */
 unsigned long  LSFAR *secondsUntilTheFeatureExpires   /* OUT */
#endif
);

/*
 * Note that the information returned by this function will be correct
```

```
 * only immediately after acquiring the handle.  The information in the
 * handle is NOT updated subsequently.
 *
 * The function is used when the clocks may not be in sync. It
 * returns the difference in seconds between the estimated current
 * time on the server and the estimated key expiration time on
 * on the server.
 *
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetKeyTimeLeftFromHandle (
#ifndef LSNOPROTO
 LS_HANDLE          lshandle,              /* IN */
 unsigned long  LSFAR *secondsUntilTheKeyExpires  /* OUT */
#endif
);

/*
 * Note that the information returned by this function will be correct
 * only immediately after acquiring the handle.  The information in the
 * handle is NOT updated subsequently.
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetLicInUseFromHandle (
#ifndef LSNOPROTO
 LS_HANDLE          lshandle,
 int        LSFAR *totalKeysIssued  /* OUT - keys issued by server */
#endif
);

/*
 * Returns the value VLS_LOCAL_UPDATE or VLS_REMOTE_UPDATE
 * depending on whether the last SUCCESSFUL update was locally done or
 * done by the SentinelLM server.
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetRenewalStatus (
#ifndef LSNOPROTO
 void
#endif
);

/*
 * Calling this function makes all future update calls
 * go directly to the SentinelLM server.
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSdisableLocalRenewal(
#ifndef LSNOPROTO
 void
```

```
#endif
);

/*
 * Calling this function allows the client libraries to process each
 * future update and send only those updates which are necessary
 * to the server. This is the default behaviour and please read the
 * user manual for further description on the default behaviour.
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSenableLocalRenewal(
#ifndef LSNOPROTO
  void
#endif
);

/*
 * This function tells us whether local renewal of keys is enabled,
 * or if all LSUpdate calls go straight to the server (disabled).
 */
VDLL32 VLS_LOC_UPD_STAT VMSWINAPI VLSisLocalRenewalDisabled(
#ifndef LSNOPROTO
  void
#endif
);

/* Function to retrieve (possibly customized) hostid of the machine */
/* THIS FUNCTION IS OBSOLETE.  Use VLSgetMachineID() instead. */
VDLL32 long VMSWINAPI VLSgetHostId(
#ifndef LSNOPROTO
void
#endif
);

/* Call this function to get a description of the client library version */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetLibInfo(
#ifndef LSNOPROTO
LS_LIBVERSION LSFAR * pInfo
#endif
);


VDLL32 long VMSWINAPI VLSgetNWerrno(
#ifndef LSNOPROTO
  void
#endif
);
```

```
VDLL32 int VMSWINAPI VLSgetServerPort(
#ifndef LSNOPROTO
  void
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSinitialize(
#ifndef LSNOPROTO
  void
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI  VLScleanup(
#ifndef LSNOPROTO
  void
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetTimeoutInterval (
#ifndef LSNOPROTO
long interval
#endif
);

VDLL32 long   VMSWINAPI VLSgetTimeoutInterval(void);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetBroadcastInterval (
#ifndef LSNOPROTO
long interval
#endif
);

VDLL32 long   VMSWINAPI VLSgetBroadcastInterval(void);

/* call the following function at the time setup is run to install
 * application using Sentinel LM
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSinstall (
#ifndef LSNOPROTO
  VLSinstallInfo LSFAR * install_info
#endif
);

/* call the following function to retrieve the information
 * which was stored at setup time
 */
```

```
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetInstallInfo (
#ifndef LSNOPROTO
  VLSinstallInfo LSFAR * install_info
#endif
);


/*-------------------------------------------------------------------------*/
/* Fingerprinting-related types and functions:                             */
/*-------------------------------------------------------------------------*/

typedef struct _vlsmachineID {
  unsigned long id_prom;            /* VLS_LOCK_ID_PROM */
  char        ip_addr[MAXLEN];      /* VLS_LOCK_IP_ADDR */
  unsigned long disk_id;            /* VLS_LOCK_DISK_ID */
  char        host_name[MAXLEN];    /* VLS_LOCK_HOSTNAME */
  char        ethernet[MAXLEN];     /* VLS_LOCK_ETHERNET */
  unsigned long nw_ipx;             /* VLS_LOCK_NW_IPX */
  unsigned long nw_serial;          /* VLS_LOCK_NW_SERIAL */
  char        portserv_addr[MAXLEN]; /* VLS_LOCK_PORTABLE_SERV */
  unsigned long custom;             /* VLS_LOCK_CUSTOM */

  unsigned long reserved;           /* For internal use */
  unsigned long unused1;            /* Reserved for future use. */
  unsigned long unused2;            /* Reserved for future use. */
} VLSmachineID;

/* Initializes a machine id struct to blank/default values. */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSinitMachineID(
#ifndef LSNOPROTO
  VLSmachineID LSFAR *machineID     /* OUT - should be pre-allocated */
#endif
);

/*
 * Sets the values of the machine id struct for the current host.
 * The input machine ID struct is initialized and then only those items
 * indicated by the lock_selector_in will (try to) be obtained and set.
 * If lock_selector_out is not NULL, *lock_selector_out is set to a bitmask
 * specifying which items could actually be obtained.
 * To try to obtain all possible machine id struct items, set
 * lock_selector_in to VLS_LOCK_ALL.
 */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSgetMachineID(
#ifndef LSNOPROTO
  unsigned long     lock_selector_in, /* IN */
```

```
 VLSmachineID  LSFAR *machineID,       /* OUT - should be pre-allocated */
 unsigned long LSFAR *lock_selector_out /* OUT - may be NULL */
#endif
);

/* Computes locking code of machineID struct based on lock selector. */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSmachineIDtoLockCode(
#ifndef LSNOPROTO
 VLSmachineID  LSFAR *machineID,    /* IN */
 unsigned long       lock_selector, /* IN */
 unsigned long LSFAR *lockCode      /* OUT - effective locking code */
#endif
);


/*-------------------------------------------------------------------------*/
/* Function Prototypes of General-Purpose Utility Functions:               */
/*-------------------------------------------------------------------------*/

/*
 * This function is called for scheduling eventhandler to be awakened after
 * so many seconds. It handles only SIGALRM signal. No. of events that can be
 * scheduled is 100. A particular eventhandler can be executed more
 * than once by specifying it in repeat_event argument. This function is
 * available only on UNIX platforms.
 */
LS_STATUS_CODE  VMSWINAPI VLSscheduleEvent(
#ifndef LSNOPROTO
unsigned long   seconds,           /* IN -- Time Interval in seconds */
void          (*eventHandler) (void),  /* IN -- Signal Handler Fn. */
long          repeat_event         /* IN -- No of event repetitions :
                                   -1 for infinite */
#endif
);

/*
 * This function is called for disabling the events scheduled using
 * VLSscheduleEvent function. To disable a particular event pass the event
 * handler function name as the argument. To disable all the events pass
 * NULL as argument. Returns LS_SUCCESS on success. This function is available
 * only on UNIX platforms.
 */
LS_STATUS_CODE  VMSWINAPI VLSdisableEvents(
#ifndef LSNOPROTO
void       (*eventHandler) (void)  /* IN -- Signal Handler Fn.: NULL for All */
#endif
```

```
);


/*-------------------------------------------------------------------*/
/* Macros with default licensing values.                             */
/* There should be no space(s) between macro name and open parenthesis.  */
/* Most of these are for backward compatibility.                     */
/*-------------------------------------------------------------------*/

#define VLS_REQUEST(feature_name, version, handle_addr) \
    LSRequest(LS_ANY,(unsigned char LSFAR *) "SentinelLM User", \
            (unsigned char LSFAR *)feature_name, \
            (unsigned char LSFAR *)version, \
            (unsigned long LSFAR *)NULL, (unsigned char LSFAR *)NULL, \
            (LS_CHALLENGE LSFAR *)NULL, handle_addr)

#define VLS_RELEASE(handle)    \
    LSRelease(handle, LS_DEFAULT_UNITS, (unsigned char LSFAR *)NULL)

#define VLS_UPDATE(handle)                          \
    LSUpdate    (handle, LS_DEFAULT_UNITS, (long LSFAR *)NULL, \
            (unsigned char LSFAR *)NULL, (LS_CHALLENGE LSFAR *)NULL)

#define VLS_INITIALIZE() VLSinitialize()

#define VLS_CLEANUP()   VLScleanup()


/*-------------------------------------------------------------------*/
/* Macros which will make all SentinelLM functions void:             */
/*-------------------------------------------------------------------*/

#ifdef NO_LICENSE
#define LSGetMessage(a1,a2,a3,a4)           (LS_SUCCESS)
#define LSRelease(a1,a2,a3)                 (LS_SUCCESS)
#define LSRequest(a1,a2,a3,a4,a5,a6,a7,a8)      (LS_SUCCESS)
#define LSUpdate(a1,a2,a3,a4,a5)            (LS_SUCCESS)
#define VLSaddFeature(a1,a2,a3)             (LS_SUCCESS)
#define VLSaddFeatureToFile(a1,a2,a3,a4)        (LS_SUCCESS)
#define VLSbatchUpdate(a1,a2,a3,a4,a5,a6,a7)    (LS_SUCCESS)
#define VLScleanup()                        (LS_SUCCESS)
#define VLSdeleteFeature(a1,a2,a3,a4)           (LS_SUCCESS)
#define VLSdisableAutoTimer(a1,a2)          (LS_SUCCESS)
#define VLSdisableLicense(a1)               (LS_SUCCESS)
#define VLSdisableLocalRenewal()            (LS_SUCCESS)
#define VLSdiscover(a1,a2,a3,a4,a5,a6,a7)       (LS_SUCCESS)
```

```
#define VLSenableLocalRenewal()              (LS_SUCCESS)
#define VLSerrorHandle(a1)                   (LS_SUCCESS)
#define VLSgetBroadcastInterval()            (9)
#define VLSgetClientInfo(a1,a2,a3,a4,a5)     (VLS_NO_MORE_CLIENTS)
#define VLSgetContactServer(a1,a2)           (LS_SUCCESS)
#define VLSgetFeatureFromHandle(a1,a2,a3)    (LS_BADHANDLE)
#define VLSgetFeatureInfo(a1,a2,a3,a4,a5)    (VLS_NO_MORE_FEATURES)
#define VLSgetFeatureTimeLeftFromHandle(a1,a2) (LS_BADHANDLE)
#define VLSgetHandleInfo(a1,a2)              (LS_BADHANDLE)
#define VLSgetHostId()                       (0)
#define VLSgetKeyTimeLeftFromHandle(a1,a2)   (LS_BADHANDLE)
#define VLSgetLibInfo(a1)                    (LS_SUCCESS)
#define VLSgetLicInUseFromHandle(a1,a2)      (LS_BADHANDLE)
#define VLSgetMachineID(a1,a2,a3)            (LS_SUCCESS)
#define VLSgetNWerrno()                      (0)
#define VLSgetRenewalStatus()                (VLS_LOCAL_UPDATE)
#define VLSgetServerList(a1,a2)              (LS_SUCCESS)
#define VLSgetServerName(a1,a2)              (LS_SUCCESS)
#define VLSgetServerNameFromHandle(a1,a2,a3) (LS_SUCCESS)
#define VLSgetServerPort()                   (5093)
#define VLSgetTimeDriftFromHandle(a1,a2)     (LS_BADHANDLE)
#define VLSgetTimeoutInterval()              (30)
#define VLSgetVersionFromHandle(a1,a2,a3)    (LS_BADHANDLE)
#define VLSgetVersions(a1,a2,a3,a4)          (VLS_NO_SUCH_FEATURE)
#define VLSinitMachineID(a1)                 (LS_SUCCESS)
#define VLSinitServerList(a1,a2)             (LS_SUCCESS)
#define VLSinitialize()                      (LS_SUCCESS)
#define VLSinstall(a1)                       (LS_SUCCESS)
#define VLSisLocalRenewalDisabled()          (VLS_LOCAL_UPD_ENABLE)
#define VLSlicense(a1,a2,a3)                 (LS_SUCCESS)
#define VLSmachineIDtoLockCode(a1,a2,a3)     (LS_SUCCESS)
#define VLSsetBroadcastInterval(a1)          (LS_SUCCESS)
#define VLSsetContactServer(a1)              (LS_SUCCESS)
#define VLSsetErrorHandler(a1,a2)            (LS_SUCCESS)
#define VLSsetRemoteRenewalTime(a1,a2,a3)    (LS_SUCCESS)
#define VLSsetServerName(a1)                 (LS_SUCCESS)
#define VLSsetTimeoutInterval(a1)            (LS_SUCCESS)
#define VLSsetTraceLevel(a1)                 (LS_SUCCESS)
#define VLSsetUserErrorFile(a1,a2)           (LS_SUCCESS)
#define VLSshutDown(a1)                      (LS_SUCCESS)
#define VLSwhere(a1,a2,a3,a4,a5,a6)          (LS_SUCCESS)
#endif /* NO_LICENSE */

#ifdef __cplusplus
}
#endif
```

```
#endif /* _LSERV_H_ */
```

```
/*****************************************************************/
/*                                                               */
/*    Copyright (C) 1998 Rainbow Technologies, Inc.              */
/*                All Rights Reserved                            */
/*                                                               */
/*   This Module contains Proprietary Information of Rainbow     */
/*  Technologies, Inc., and should be treated as Confidential.   */
/*****************************************************************/

/*H*****************************************************************
* FILENAME    : lservcst.h
*
* DESCRIPTION :
*      This file contains prototypes and header declarations for
*      customizing the client/server. There are various aspects of the
*      client/server that can be customized to suit a vendor's needs.
*      This file lists all the aspects.
*
* USAGE       :
*      All files related to customization must include this file.
*      IMPORTANT- If a vendor customizes his/her server in any way,
*      he/she must also change the port number of his/her server via
*      the API call VLSchangePortNumber(), so that the customized
*      server does not interfere with other vendors' applications that
*      rely on a default (uncustomized) server. Of course, the clients
*      must also be modified to contact the server on the new port
*      number, using the client API call VLSsetServerPort().
*
* NOTES       :
*      All functions in this file that are marked OVERRIDE, when present
*      in vendor's object files, will override default function bodies
*      present in static libraries of SentinelLM. For this to work
*      correctly, vendor must specify his/her overriding object files
*      BEFORE SentinelLM libraries in the linker command. These
*      functions are called by the client/server as and when needed.
*
*      All functions in this file that are marked BUILT-IN, are
*      functions that can be called from any vendor functions. Vendor
*      should NOT override these functions (i.e., provide his/her own
*      functions by the same names).
*
*      All functions in this file should be customized in the same
*      manner in the standalone library as well, if the standalone
*      library is being used.
*
*H*/
```

```
#ifndef _LSERVCST_H_
#define _LSERVCST_H_


#ifdef __cplusplus
 extern "C" {
#endif

#include "lserv.h"  /* client API, needed for common typedefs */


/*--------------------------------------------------------------------*/
/* Prototypes for client-side customization:                          */
/*--------------------------------------------------------------------*/

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetHoldTime (  /* BUILT-IN */
#ifndef LSNOPROTO
  unsigned char   LSFAR *feature_name, /* IN */
  unsigned char   LSFAR *version,     /* IN */
  int             hold_time     /* IN */
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetSharedId (  /* BUILT-IN */
#ifndef LSNOPROTO
  int shared_id,                                  /* IN */
  LS_STATUS_CODE (VMSWINAPI * mySharedIdFunc) (char LSFAR *)  /* IN */
#endif
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetSharedIdValue(  /* BUILT-IN */
#ifndef LSNOPROTO
  int             shared_id,    /* IN */
  char       LSFAR *sharedIdValue  /* IN */
#endif
);

VDLL32 void VMSWINAPI VLSsetServerPort(  /* BUILT-IN */
#ifndef LSNOPROTO
  int port_number               /* IN */
#endif
);


/*--------------------------------------------------------------------*/
```

```
/* Prototypes for client-and-server-side customization:              */
/*-------------------------------------------------------------------*/

/* Supply custom hostid function/mechanism */
VDLL32 LS_STATUS_CODE VMSWINAPI VLSsetHostIdFunc(  /* BUILT-IN */
#ifndef LSNOPROTO
 long (VMSWINAPI * customGetHostIdFunc)(void)  /* IN */
#endif
);


/* Network messages encryption/decryption customization: */
int VLSencryptMsg(                  /* OVERRIDE */
#ifndef LSNOPROTO
 char *decrypted_mesg,        /* IN */
 char *encrypted_mesg,        /* OUT - allocated by caller */
 int  size                /* IN */
#endif
);

int VLSdecryptMsg(                  /* OVERRIDE */
#ifndef LSNOPROTO
 char *encrypted_mesg,        /* IN */
 char *decrypted_mesg,        /* OUT - allocated by caller */
 int  size                /* IN */
#endif
);


/*-------------------------------------------------------------------*/
/* Types and prototypes for server hook functions customization:     */
/*-------------------------------------------------------------------*/

#define HOOK_LS_MAX_PATHLEN        128  /* Path Length */
#define HOOK_CLIENT_IDENTIFIER_SIZE  100  /* Client identifier size */

/* ***** Event types ***** */
#define LS_REQ_PRE          0 /* EVENT : Before processing lsreq() */
#define LS_REQ_POST          1 /* EVENT : After  processing lsreq() */
#define LS_REL_PRE          2 /* EVENT : Before processing lsrel() */
#define LS_REL_POST          3 /* EVENT : After  processing lsrel() */

/* ***** Error codes on server side ***** */
#define LSERV_STATUS_SUCCESS      LS_SUCCESS   /* Success status */
#define LSERV_STATUS_DENY        101  /* Denial by vendor event handler */
```

```
typedef int LSERV_STATUS;

/* Structure for File Location Info passed to vendor event handlers. */
typedef struct {
 char     lservrcFile   [HOOK_LS_MAX_PATHLEN];  /* lservrc file path */
 char     lservrcCnfFile [HOOK_LS_MAX_PATHLEN];  /* lserv cnf file path */
 char     lservStaFile  [HOOK_LS_MAX_PATHLEN];  /* lserv stat file path */
 char     lservLogFile  [HOOK_LS_MAX_PATHLEN];  /* lserv log file path */
 char     lsGrResvFile  [HOOK_LS_MAX_PATHLEN];  /* lserv group file path */
 char     reserved      [HOOK_LS_MAX_PATHLEN];  /* reserved */
} VLSfileLocInfo;

/* Structure for Misc. Info passed to vendor event handlers. */
typedef struct {
 char        ipAddress    [MAX_NAME_LEN];   /* of client */
            /* Flags indicate status of tests for this request: */
 int         nodeLockPass;      /* 1 => Node locking tests pass */
 int         siteLicensePass;   /* 1 => Site licensing tests pass */
 int         licExpirationPass; /* 1 => License expiration tests pass */
 int         clockTamperPass;   /* 1 => Clock tampering tests pass */
 char        reserved     [MAX_NAME_LEN];
} VLSmiscInfo;

/* The complete structure passed to vendor event handlers. */
typedef struct {
 VLSclientInfo   clientInfoStruct; /* Same as client API struct */
 VLSfeatureInfo  featureInfoStruct; /* Same as client API struct */
 VLSfileLocInfo  fileLocInfoStruct;
 VLSmiscInfo     miscInfoStruct;
} VLShandlerStruct;


/*
 * Called by server during server initialization. This is where
 * calls to VLSeventAddHook() should be placed, to configure the server
 * to consult vendor event handler functions.
 */
LSERV_STATUS VLSserverVendorInitialize(   /* OVERRIDE */
#ifndef LSNOPROTO
 void
#endif
);

/* Call to register an event handler with the server. */
#ifndef _VWIN31X_
LSERV_STATUS VLSeventAddHook(          /* BUILT-IN */
```

```
#ifndef LSNOPROTO
 int     eventName,  /* IN - event type , LS_REQ / LS_REL */
                              /* _PRE / _POST */
 int     (*handlerFuncPtr)(VLShandlerStruct *, char *, char *, int),
              /* IN - function pointer */
 char  *  identifier  /* IN - client identifier to match */
#endif
);
#endif

/*-------------------------------------------------------------------------*/
/* Client-side calls to use Server Hooks:                                  */
/*-------------------------------------------------------------------------*/

/* Struct passed to server from client while using server hooks: */
typedef  struct {
  char  identifier[MAX_NAME_LEN];
  char  inBuf[MAX_BUF_LEN];  /* String passed to the server */
  char  outBuf[MAX_BUF_LEN]; /* String returned by the server */
} VLSserverInfo;


VDLL32 LS_STATUS_CODE VMSWINAPI VLSinitServerInfo (  /* BUILT-IN */
#ifndef LSNOPROTO
 VLSserverInfo   LSFAR   *serverInfo  /* OUT - allocated by caller */
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSrequestExt (  /* BUILT-IN */
#ifndef LSNOPROTO
 unsigned char  LSFAR *license_system,  /* IN */
 unsigned char  LSFAR *publisher_name,  /* IN */
 unsigned char  LSFAR *product_name,    /* IN */
 unsigned char  LSFAR *version,         /* IN */
 unsigned long  LSFAR *units_reqd,      /* IN */
 unsigned char  LSFAR *log_comment,     /* IN */
 LS_CHALLENGE   LSFAR *challenge,       /* INOUT - allocated by caller */
 LS_HANDLE      LSFAR *lshandle,        /* OUT - allocated by caller */
 VLSserverInfo  LSFAR *serverInfo       /* INOUT - allocated by caller */
#endif /* LSNOPROTO */
);

VDLL32 LS_STATUS_CODE VMSWINAPI VLSreleaseExt (  /* BUILT-IN */
#ifndef LSNOPROTO
 LS_HANDLE          lshandle,       /* IN */
 unsigned long      unused1,
```

```
  unsigned char  LSFAR *log_comment,    /* IN */
  VLSserverInfo  LSFAR *serverInfo      /* INOUT - allocated by caller */
#endif /* LSNOPROTO */
);


/*-----------------------------------------------------------------------*/
/* Time tamper customization (for server and standalone mode):           */
/*-----------------------------------------------------------------------*/

typedef enum {VLS_CONT_AFTER_TM_TAMPER, VLS_EXIT_AFTER_TM_TAMPER}
      VLSactionOnTmTamper;

typedef enum {VLS_ENABLE_DEFAULT_TM_TAMPER,
VLS_DISABLE_DEFAULT_TM_TAMPER}
      VLStmTamperMethod;


/*
 * Called by server each time server needs to verify whether the system
 * clock has been set back.  Default behavior of the server can be
 * customized here.  Note this is called BEFORE any checks are performed
 * by the server.
 */
void VLSconfigureTimeTamper (           /* OVERRIDE */
#ifndef LSNOPROTO
  VLSactionOnTmTamper *   actionOnTmTamper,     /* OUT */
  VLStmTamperMethod   *   tmTamperMethod,       /* OUT */
  long                *   gracePeriod,          /* OUT */
  int                 *   percentViolations,    /* OUT */
  int                 *   numViolationsForError  /* OUT */
#endif
);

/*
 * Vendor's function to tell the server if clock has been set back.
 * Called only in case vendor's VLSconfigureTimeTamper() function returns
 * tmTamperMethod to be VLS_DISABLE_DEFAULT_TM_TAMPER, not otherwise.
 * Should return 0 if clock is not set back.
 */
int VLSisClockSetBack(void);                /* OVERRIDE */


/*-----------------------------------------------------------------------*/
/* License encryption/decryption customization (server and standalone)   */
/*-----------------------------------------------------------------------*/
```

```
int VLSencryptLicense(                  /* OVERRIDE */
#ifndef LSNOPROTO
 char *decrypted_mesg,          /* IN */
 char *encrypted_mesg,          /* OUT - allocated by caller */
 int  size                  /* IN */
#endif
);

int VLSdecryptLicense(                  /* OVERRIDE */
#ifndef LSNOPROTO
 char *encrypted_mesg,          /* IN */
 char *decrypted_mesg,          /* OUT - allocated by caller */
 int  size                  /* IN */
#endif
);


/*---------------------------------------------------------------------*/
/* Server UDP port number customization:                               */
/*---------------------------------------------------------------------*/

/* Should return the desired UDP port number of server */
int VLSchangePortNumber(                /* OVERRIDE */
#ifndef LSNOPROTO
 int currentPort     /* IN  - Currently configured port number */
#endif
);



/*---------------------------------------------------------------------*/
/* Macros which will make all SentinelLM functions void:               */
/*---------------------------------------------------------------------*/

#ifdef NO_LICENSE
#define VLSinitServerInfo(a1)                   (LS_SUCCESS)
#define VLSeventAddHook(a1,a2,a3)               (LS_SUCCESS)
#define VLSreleaseExt(a1,a2,a3,a4)              (LS_SUCCESS)
#define VLSrequestExt(a1,a2,a3,a4,a5,a6,a7,a8,a9) (LS_SUCCESS)
#define VLSsetHoldTime(a1,a2)                   (LS_SUCCESS)
#define VLSsetHostIdFunc(a1)                    (LS_SUCCESS)
#define VLSsetServerPort(a1)                    /* void return */
#define VLSsetSharedId(a1,a2)                   (LS_SUCCESS)
#define VLSsetSharedIdValue(a1,a2)              (LS_SUCCESS)
#endif
```

```
#ifdef __cplusplus
}
#endif


#endif /* _LSERVCST_H_ */
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <1999-09-21 10:26:16 ayahil>

PROGRAM Main

 USE Dflib, ONLY: Getarg
 USE Dfport, ONLY: Dtime, Iargc, Time
 USE Interfaces, ONLY: Algebron, Gen_dev, Prepare, Sample
 USE Interfaces, ONLY: Get_gof
 USE Nrtype
 USE Ran_state, ONLY: Ran_seed
 USE Utils, ONLY: Bootstrap, Indgen
 IMPLICIT NONE
                    ! Locals
 CHARACTER, ALLOCATABLE, DIMENSION(:) :: arg*3, date*8
 INTEGER :: add_iter, cntmn, i, iarg, iboot, isim, mp, n, narg, nboot, &
    & nprob, nsim, p, p0, pstep, prn, seed, stepmx
 INTEGER, ALLOCATABLE, DIMENSION(:) :: idx, freq
 LOGICAL, ALLOCATABLE, DIMENSION(:,:) :: msk
 REAL :: ta(2)
 REAL(sp) :: abstol, gof, gof1, gof2, gof_ev, gof_std, reltol, nsig, &
    & prob_avg, prob_var, prune, signif, tau
 REAL(sp), ALLOCATABLE, DIMENSION(:) :: chi2, prob, ravg, ravg_t, w, w0, z
 REAL(sp), ALLOCATABLE, DIMENSION(:,:) :: r, r_t, s, stot, v, v0, v1, v2, &
    & vtot, x, x_t
                    ! Runtime arguments
 narg = Iargc()
 ALLOCATE( arg(MAX(narg,1)) )
 arg(1) = 'bin'
 DO iarg = 1,narg
   CALL Getarg( iarg, arg(iarg) )
 END DO
                    ! Read in sample data
 IF( arg(1) == 'asc' .OR. arg(1) == 'ASC' ) THEN
   OPEN( 7, FILE='../data/amos.data', STATUS='old' )
   OPEN( 8, FILE='../data/amos.bin', FORM='unformatted', STATUS='unknown' )
   READ(7,*) n, mp
   WRITE(*,*) n, mp
   WRITE(8) n, mp
   ALLOCATE( date(n), x_t(n,mp), w(n) )
   DO i = 1,n
     READ(7,*) date(i), x_t(i,:)
```

```
  END DO
  WRITE(8) date, x_t
  CLOSE(8)
ELSE
  OPEN( 7, FILE='../data/amos.bin', FORM='unformatted', STATUS='old' )
  READ(7) n, mp
  ALLOCATE( date(n), x_t(n,mp), w(n) )
  READ(7) date, x_t
  CLOSE( 7 )
END IF
DEALLOCATE( arg, date )
WRITE(*,'(a,f10.3)') 'Data input:', Dtime( ta )
                    ! Choose computation parameters
WRITE(*,'(a,$)') 'Enter cntmn, p0, prune, seed, tau: '
READ(*,*)           cntmn, p0, prune, seed, tau
IF( seed == 0 ) THEN
  CALL Ran_seed( Time() )
ELSE
  CALL Ran_seed( seed )
END IF
IF( p0 <= 0 .OR. p0 > mp ) p0 = mp
pstep = mp/p0
ALLOCATE( x(n,p0) )
x = x_t(:,pstep:p0*pstep:pstep)
DEALLOCATE( x_t )
ALLOCATE( idx(p0), r_t(n,p0), ravg_t(p0) )
w = EXP( (Indgen(n) - n)/tau )
                    ! Prepare daily returns
CALL Prepare( cntmn, idx, n, p, p0, prune, r_t, ravg_t, w, x )
ALLOCATE( r(n,p), ravg(p) )
r = r_t(:,idx(:p))
ravg = ravg_t(idx(:p))
DEALLOCATE( idx, r_t, ravg_t, x )
ALLOCATE( chi2(n), prob(n), v(p,p), z(p*(p+1)) )
WRITE(*,'(a,2i5,f10.2,1p3e15.7)') 'Initialization:', p0, p, &
   & MINVAL( r, MASK = r /= -999.0_sp ), &
   & MAXVAL( r, MASK = r /= -999.0_sp )
                    ! Algebron Covariance estimate
WRITE(*,'(a,$)') 'Enter abstol, nboot, nsig, nsim, reltol, signif, stepmx: '
READ(*,*)           abstol, nboot, nsig, nsim, reltol, signif, stepmx
add_iter = 1
prn = 1
z = 0.0_sp
CALL Algebron( abstol, add_iter, chi2, gof, gof_ev, gof_std, n, nsig, p, &
   & prn, prob, r, reltol, signif, stepmx, v, w, z )
nprob = COUNT( prob /= 0.0_sp )
```

```
prob_avg = SUM( prob, MASK= prob /= 0.0_sp )/nprob
prob_var = (SUM( prob**2, MASK= prob /= 0.0_sp ) - nprob*prob_avg**2) &
   & /(nprob - 1)
WRITE(*,*) 'Gof, gof_ev, gof_std, prob_avg, prob_var:', &
   & gof, gof_ev, gof_std, prob_avg, prob_var, 1.0_sp/12.0_sp, &
   & 1.0_sp/SQRT( 12.0_sp*nprob )
add_iter = 0
                    ! Update computation (only if no bootstrap
                    ! computation is done)
IF( nboot <= 1 ) THEN
  add_iter = 0
  CALL Algebron( abstol, add_iter, chi2, gof, gof_ev, gof_std, n, nsig, p, &
     & prn, prob, r, reltol, signif, stepmx, v, w, z )
  nprob = COUNT( prob /= 0.0_sp )
  prob_avg = SUM( prob, MASK= prob /= 0.0_sp )/nprob
  prob_var = (SUM( prob**2, MASK= prob /= 0.0_sp ) - nprob*prob_avg**2) &
     & /(nprob - 1)
  WRITE(*,*) 'Gof, gof_ev, gof_std, prob_avg, prob_var:', &
     & gof, gof_ev, gof_std, prob_avg, prob_var, 1.0_sp/12.0_sp, &
     & 1.0_sp/SQRT( 12.0_sp*nprob )
END IF
                    ! Bootstraps
IF( nboot > 1 ) THEN
  add_iter = 1
  ALLOCATE( freq(n), v0(p,p), v1(p,p), v2(p,p), w0(n) )
  v0 = v
  v1 = 0.0_sp
  v2 = 0.0_sp
  w0 = w
  DO iboot = 1,nboot
    CALL Bootstrap( freq, n, seed )
    w = w0*freq
    z = 0.0_sp
    WRITE(*,*) iboot
    CALL Algebron( abstol, add_iter, chi2, gof, gof_ev, gof_std, n, nsig, &
       & p, prn, prob, r, reltol, signif, stepmx, v, w, z )
    v1 = v1 + v
    v2 = v2 + v**2
  END DO
                    ! Bootstrap statistics
  v1 = v1/nboot
  v2 = SQRT( (v2 - nboot*v1**2)/(nboot - 1) )
  WRITE(*,'(1p5e15.5)') v0
  WRITE(*,*)
  WRITE(*,'(1p5e15.5)') v1
  WRITE(*,*)
```

```
  WRITE(*,'(1p5e15.5)') v2
  WRITE(*,*)
  WRITE(*,'(1p5e15.5)') v1 - v0
  WRITE(*,*)
  WRITE(*,'(1p5e15.5)') (v1 - v0)/(v2 + TINY(v2))
  WRITE(*,*)
                    ! Cleanup
  v = v0
  w = w0
  DEALLOCATE( freq, v0, v1, v2, w0 )
END IF
                    ! Simulations with the same covariance matrix
IF( nsim > 1 ) THEN
  ALLOCATE( msk(n,p), v0(p,p), v1(p,p), v2(p,p) )
  gof1 = 0.0_sp
  gof2 = 0.0_sp
  msk = ( r == -999.0_sp )
  v0 = v
  v1 = 0.0_sp
  v2 = 0.0_sp
  Simulation_loop: DO isim = 1,nsim
    CALL Gen_dev( v0, n, p, r )
    WHERE( msk )
      r = -999.0_sp
    END WHERE
    gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v0, w )
    WRITE(*,*) 'Simulation gof:', isim, gof, gof_ev, gof_std, &
       & (gof - gof_ev)/gof_std
    gof1 = gof1 + gof
    gof2 = gof2 + gof**2
                    ! Algebron method with the simulated data
    add_iter = 1
    z = 0.0_sp
    WRITE(*,*) isim
    CALL Algebron( abstol, add_iter, chi2, gof, gof_ev, gof_std, n, nsig, &
       & p, prn, prob, r, reltol, signif, stepmx, v, w, z )
    v1 = v1 + v
    v2 = v2 + v**2
  END DO Simulation_loop
                    ! Statistics of simulations
  gof1 = gof1/nsim
  gof2 = SQRT( (gof2 - nsim*gof1**2)/(nsim - 1) )
  v1 = v1/nsim
  v2 = SQRT( (v2 - nsim*v1**2)/(nsim - 1) )
  WRITE(*,'(1p5e15.5)') v0
  WRITE(*,*)
```

[main.f90]

```
    WRITE(*,'(1p5e15.5)') v1
    WRITE(*,*)
    WRITE(*,'(1p5e15.5)') v2
    WRITE(*,*)
    WRITE(*,'(1p5e15.5)') v1 - v0
    WRITE(*,*)
    WRITE(*,'(1p5e15.5)') (v1 - v0)*SQRT( REAL(nsim) )/(v2 + TINY(v2))
    WRITE(*,*)
    WRITE(*,*) gof1, gof_ev, gof2, gof_std, (gof1 - gof_ev)*SQRT( REAL(nsim) ) &
      & /gof_std
                    ! Cleanup
    DEALLOCATE( msk, v0, v1, v2 )
  END IF
                    ! Cleanup
  DEALLOCATE( chi2, prob, r, ravg, v )
                    ! Done
  WRITE(*,'(a,f10.3)') 'Computation:', Dtime( ta )
END PROGRAM Main
```

```
                    /* ***************************************
                    /*  Copyright (C) 1998 by Algebron LLC.
                    /*  All rights reserved.
                    /*  Unauthorized reproduction prohibited.
                    /* *************************************** */
                    /* Time-stamp: <1999-09-21 10:27:53 ayahil> */
/*

  Sample program to demonstrate the use of the Fortran program ALGEBRON and
  auxiliary modules.  PLEASE NOTE:

  1.  FORTRAN PROGRAMS NEED VARIABLES BY ADDRESS.  a variable has to be
  preceded by & to pass an address.  However, pass a pointer as is, since it
  is already an address.

  2.  C names of Fortran modules are system-dependent.  On Sun Solaris add an
  underline (_) at the end of the name.

  */

#include <stdio.h>
#include <math.h>
#include "algebron_all.h"
#define DATA_FILENAME "../data/amos.data"
#define DATE_STRING_SIZE       10

main()
{
                    /* Data types */
  char  date[DATE_STRING_SIZE];
  int add_iter, cntmn, mp, n, p, p0, prn, pstep, stepmx, update; /* scalar */
  int *idx;                 /* 1-d */
  double abstol, gof, gof_ev, gof_std, nsig, prune, reltol, signif,
    tau;                  /* scalar */
  double *chi2, *prob, *ravg_t, *w, *z; /* 1-d */
  double *r, *r_n, *r_t, *v, *x, *x_t; /* 2-d */
                    /* Data File Pointer */
  FILE * dfp;
                    /* Data types of auxiliary variables */
  float f1,f2,f3,f4;
  int i, i1, j, k;
                    /* Read in sample data */
  dfp = fopen( DATA_FILENAME, "r" );
  fscanf( dfp, "%d %d", &n, &mp );
  printf( "n, mp: \t%d\t%d\n", n, mp );
  chi2 = (double*) malloc( sizeof( double ) * n );
```

```
prob = (double*) malloc( sizeof( double ) * n );
w = (double*) malloc( sizeof( double ) * n );
x_t = (double*) malloc( sizeof( double ) * n * mp );
                        /* Read the dates and prices */
for( i = 0; i<n; i++ )
 {
  fscanf( dfp, "%s", date ); /* Read and throw away*/
  for( j=0; j<mp; j++ )
   {
    fscanf( dfp, "%f\t", &f1 ); /* Read and save */
    x_t[j*n + i] = (double)f1;
   }
 }
                        /* Enter parameters for selection of data
                           subset, PREPARE parameters, and
                           exponentially backward declining weights */
printf( "Enter cntmn p0 prune tau: ");
scanf( "%d %d %f %f", &cntmn, &p0, &f1, &f2 );
prune = (double)f1;
tau = (double)f2;
if( (p0 <=0) || (p0 > mp ) ) p0 = mp;
pstep = mp/p0;
if( tau == 0.0 ) tau = 1e20;
                        /* Create a subset of the data */
x = (double*) malloc ( sizeof( double ) * n * p0);
for( j=pstep-1, k=0; j<p0*pstep; j+=pstep, k++ )
 for( i=0; i < n; i++ )
  x[n*k+i] = x_t[n*j+i];
free( x_t );
                        /* Exponentially backward declining weights */
for( i=0; i<n; i++ )
 w[i] = exp( ((double)(i-n+1))/tau );
                        /* Memory allocation for PREPARE variables */
idx = (int*) malloc ( sizeof( int ) * p0);
r_t= (double*) malloc ( sizeof( double ) * n * p0);
ravg_t= (double*) malloc ( sizeof( double ) * p0);
                        /* PREPARE the data, rejecting securities which
                           do no meet inclusion criteria */
prepare_( &cntmn, idx, &n, &p, &p0, &prune, r_t, ravg_t, w, x );
                        /* Need to decrement the indices idx by one to
                           bring Fortran indices (1,...,n) to C
                           notation (0,...,n-1) */
for( i=0; i<p; i++ )
 idx[i]--;
                        /* Restrict ALGEBRON variables to the subsets
                           corresponding to the valid securities */
```

```
r= (double*) malloc ( sizeof( double ) * n * p);
for (j = 0; j<p; j++)
  for (i = 0; i<n; i++)
    r[n*j+i] = r_t[n*idx[j]+i];
                                /* Free unneeded variables */
free( idx );
free( r_t );
free( ravg_t );
free( x );
                        /* Additional variables needed by ALGEBRON */
v= (double*) malloc ( sizeof( double ) * p * p);
z= (double*) malloc ( sizeof( double ) * p * (p+1));
                        /* Initialize z to zero */
for (i = 0; i<(p*(p+1)); i++)
  z[i]=0.0;
                        /* Read in most ALGEBRON control parameters,
                          Use defaults for others.  See ALGEBRON.H */
printf("Enter abstol nsig reltol signif stepmx: ");
scanf("%f %f %f %f %d", &f1, &f2, &f3, &f4, &i1);
abstol = (double) f1;
nsig = (double) f2;
reltol = (double) f3;
signif = (double) f4;
stepmx = i1;
                        /* ALGEBRON covariance estimate */
algebron_( &abstol, &add_iter_def, chi2, &gof, &gof_ev, &gof_std, &n,
          &nsig, &p, &prn_def, prob, r, &reltol, &signif,
          &stepmx, v, w, z );
                        /* Print the chi-squared */
printf( "gof, gof_ev, gof_std: %f %f %f\n", gof, gof_ev, gof_std );
                        /* Update the ALGEBRON covariance estimate */
add_iter = 0;
algebron_( &abstol, &add_iter, chi2, &gof, &gof_ev, &gof_std, &n, &nsig, &p,
          &prn_def, prob, r, &reltol, &signif, &stepmx, v, w, z );
                        /* Print the chi-squared */
printf( "gof, gof_ev, gof_std: %f %f %f\n", gof, gof_ev, gof_std );
                        /* Generate fake returns with the covariance
                          matrix found by ALGEBRON */
r_n = (double*) malloc( sizeof( double ) * n * p );
gen_dev_( v, &n, &p, r_n );
for( j=0; j<p; j++ )
  for( i=0; i<n; i++ )
    if( ! ( r[n*j+i] == -999.0 ) ) r[n*j+i] = r_n[n*j+i];
free( r_n );
                        /* Reinitialize z */
for (i = 0; i<(p*(p+1)); i++)
```

```
  z[i]=0.0;
 algebron_( &abstol, &add_iter_def, chi2, &gof, &gof_ev, &gof_std, &n, &nsig,
            &p, &prn_def, prob, r, &reltol, &signif, &stepmx, v, w, z );
                        /* Print the chi-squared */
 printf( "gof, gof_ev, gof_std: %f %f %f\n", gof, gof_ev, gof_std );

}
```

```
#**     Time-stamp: <1999-09-21 10:29:06 ayahil>
# C & FORTRAN compilation & linking flags
R8=    -r8
CFLAGS= $(COPTS) $(ENDIAN)
FFLAGS=      $(FOPTS) $(ENDIAN)

# Local subroutine library
SBR=  libsbr$(R8)-$(HOSTTYPE).a

# Library link symbols
DXML=        -ldxml
IMSL=        -limsl
FITSIO=      -lfitsio
LIB=   -llib$(R8)-$(HOSTTYPE)
MINUIT=      -lminuit
MINUITL=-lminuitl
MONGO=       -lmongo
MYNR=        -lmyNR$(R8)-$(HOSTTYPE)
NAG= -lnag$(R8)
NR=    -lNR$(R8)
SM=    -lparser -lplotsub -ldevices -lutils -lX11

# Library path (/usr/lib is searched anyway, but /usr/local/lib not always)
LPATH=       -L$(HOME)/lib -L/usr/local/lib

# Libraries to be linked (order is important)
FLIBS=       $(SBR)

# Modules
NRMODS=    \
        nrtype.mod              \
        nr.mod                          \
        nrutil.mod              \
        mynr.mod                \
        ran_state.mod           \
        my_ran_state.mod        \

MODS=        \
        dflib.mod               \
        dfport.mod              \
        interfaces.mod                  \
        parm.mod                \
        utils.mod               \

# Object codes of subroutines (to be placed in SBR)
OBJS=        \
```

```
	$(SBR)(algebron.o)		\
	$(SBR)(brent.o)			\
	$(SBR)(bootstrap.o)		\
	$(SBR)(choldc.o)		\
	$(SBR)(cholinv.o)		\
	$(SBR)(cholsl.o)		\
	$(SBR)(cml.o)			\
	$(SBR)(covar.o)			\
	$(SBR)(cull.o)			\
	$(SBR)(dmatmul.o)		\
	$(SBR)(eigsrt.o)		\
	$(SBR)(frprmn.o)		\
	$(SBR)(func.o)			\
	$(SBR)(gammln.o)		\
	$(SBR)(gammq.o)			\
	$(SBR)(gasdev.o)		\
	$(SBR)(gcf.o)			\
	$(SBR)(gen_covar.o)		\
	$(SBR)(gen_dev.o)		\
	$(SBR)(get_gof.o)		\
	$(SBR)(gser.o)			\
	$(SBR)(indexx.o)		\
	$(SBR)(indgen.o)		\
	$(SBR)(linmin.o)		\
	$(SBR)(ml.o)			\
	$(SBR)(mnbrak.o)		\
	$(SBR)(my_ran1.o)		\
	$(SBR)(next.o)			\
	$(SBR)(permute.o)		\
	$(SBR)(prepare.o)		\
	$(SBR)(pythag.o)		\
	$(SBR)(ran1.o)			\
	$(SBR)(sample.o)		\
	$(SBR)(sort.o)			\
	$(SBR)(spd.o)			\
	$(SBR)(tqli.o)			\
	$(SBR)(trace.o)			\
	$(SBR)(tred2.o)			\
	$(SBR)(wher.o)			\

# Object codes of main programs (kept in the directory)
PGMS=		\
	main.o			\
	test_func.o		\
	test_gen_covar.o	\
	test_gen_dev.o		\
```

```
        test_gof.o                      \
        test_spd.o                      \
        test_wher.o                     \

# Update SBR & PGMS (default is SBR only; make compile for both)
$(SBR):                 $(NRMODS) $(MODS) $(OBJS)
        ranlib $@
$(PGMS):        $(NRMODS) $(MODS)
compile:        $(SBR) $(PGMS)

# Executables
$(PGMS:.o=):            $(SBR) $(PGMS)
        -$(FC) $@.o $(LPATH) $(FLIBS) $(LDFLAGS) -o $@ || rm $@
all:    $(PGMS:.o=)

# Nonautomatic dependencies (e.g., .mod files must be set in the correct order)
interfaces.mod:         parm.mod

# Clean the directory
clean:
        -rm *.o $(PGMS:.o=) mon.out
        -precision "$(R8)"
veryclean:      clean
        -rm $(NRMODS) $(MODS) $(SBR)

# Print updated .f* files
print:  print-stamp

print-stamp:    *.$(FC:77=)
        @-echo $?
        @-prl `match "$?" $(PGMS:.o=.$(FC:77=))`        \
                `nomatch "$?" $(PGMS:.o=.$(FC:77=)) | \
                grep -v '^f2ctmp_'` && touch $@ &

# Print everything
printall:
        @-rm print-stamp
        @-make print

# Prevent files from being erased by job interruption
.PRECIOUS:  $(SBR) $(PGMS) $(PGMS:.o=)

# Add to the suffix list
.SUFFIXES:  .f90

# Compilation rules
```

```
.c.o:
	$(CC) $(CFLAGS) -c $<
.f90.o:
	$(FC) $(FFLAGS) -c $<

# Prepare module files
.f.mod .f90.mod:
	rmlib $(SBR) $(@:.mod=)
	rmobj $(@:.mod)
	$(FC) $(FFLAGS) -c $<
	@ar ruv $(SBR) $*.o
	@rm $*.o

# Library rules
.c.a:
	$(CC) $(CFLAGS) -c $<
	@ar ruv $@ $*.o
	@rm $*.o
.f.a .f90.a:
	$(FC) $(FFLAGS) -c $<
	@ar ruv $@ $*.o
	@rm $*.o
```

```
#**     Time-stamp: <1999-09-21 10:29:44 ayahil>
# C & FORTRAN compilation & linking flags
R8=    -r8
CFLAGS= $(COPTS) $(ENDIAN)
FFLAGS=     $(FOPTS) $(ENDIAN)

# Local subroutine library
SBR=  libsbr$(R8)-$(HOSTTYPE).a

# Library link symbols
DXML=       -ldxml
IMSL=       -limsl
FITSIO=     -lfitsio
LIB=  -llib$(R8)-$(HOSTTYPE)
MINUIT=     -lminuit
MINUITL=-lminuitl
MONGO=      -lmongo
MYNR=       -lmyNR$(R8)-$(HOSTTYPE)
NAG= -lnag$(R8)
NR=   -lNR$(R8)
SM=   -lparser -lplotsub -ldevices -lutils -lX11

# Library path (/usr/lib is searched anyway, but /usr/local/lib not always)
LPATH=      -L$(HOME)/lib  -L../lib -L/usr/local/lib

# Libraries to be linked (order is important)
FLIBS=      $(SBR) -lls -lnonet -lnsl -lsocket

# Modules
NRMODS= \
        nrtype.M                \
        nr.M                    \
        nrutil.M                \
        mynr.M                      \
        ran_state.M             \
        my_ran_state.M              \

MODS=       \
        dflib.M                     \
        dfport.M                \
        interfaces.M            \
        parm.M                      \
        utils.M                     \

# Object codes of subroutines (to be placed in SBR)
OBJS=       \
```

```
	$(SBR)(algebron_sec.o)		\
	$(SBR)(brent.o)			\
	$(SBR)(bootstrap.o)		\
	$(SBR)(choldc.o)		\
	$(SBR)(cholinv.o)		\
	$(SBR)(cholsl.o)		\
	$(SBR)(cml.o)			\
	$(SBR)(covar.o)			\
	$(SBR)(cull.o)			\
	$(SBR)(dmatmul.o)		\
	$(SBR)(eigsrt.o)		\
	$(SBR)(frprmn.o)		\
	$(SBR)(func.o)			\
	$(SBR)(gammln.o)		\
	$(SBR)(gammq.o)			\
	$(SBR)(gasdev.o)		\
	$(SBR)(gcf.o)			\
	$(SBR)(gen_covar.o)		\
	$(SBR)(gen_dev.o)		\
	$(SBR)(get_gof.o)		\
	$(SBR)(gser.o)			\
	$(SBR)(indexx.o)		\
	$(SBR)(indgen.o)		\
	$(SBR)(linmin.o)		\
	$(SBR)(ml.o)			\
	$(SBR)(mnbrak.o)		\
	$(SBR)(my_ran1.o)		\
	$(SBR)(next.o)			\
	$(SBR)(permute.o)		\
	$(SBR)(prepare.o)		\
	$(SBR)(pythag.o)		\
	$(SBR)(ran1.o)			\
	$(SBR)(sample.o)		\
	$(SBR)(sentinel_wrapper.o)	\
	$(SBR)(sort.o)			\
	$(SBR)(spd.o)			\
	$(SBR)(tqli.o)			\
	$(SBR)(trace.o)			\
	$(SBR)(tred2.o)			\
	$(SBR)(wher.o)			\

# Object codes of main programs (kept in the directory)
PGMS=		\
	main.o			\
	mainc.o			\
	test_func.o		\
```

```
        test_gen_covar.o        \
        test_gen_dev.o          \
        test_gof.o              \
        test_spd.o              \
        test_wher.o             \

# Update SBR & PGMS (default is SBR only; make compile for both)
$(SBR):         $(NRMODS) $(MODS) $(OBJS)
        ranlib $@
$(PGMS):    $(NRMODS) $(MODS)
compile:    $(SBR) $(PGMS)

# Executables
$(PGMS:.o=):        $(SBR) $(PGMS)
        -$(FC) $@.o $(LPATH) $(FLIBS) $(LDFLAGS) -o $@ || rm $@
all:    $(PGMS:.o=)

# Nonautomatic dependencies (e.g., .M files must be set in the correct order)
interfaces.M:  parm.M
msc.o:         algebron.h gen_dev.h prepare.h
sentinel_wrapper.c: lserv.h lservcst.h

# Clean the directory
clean:
        -rm *.o $(PGMS:.o=) mon.out
        -precision "$(R8)"
veryclean:    clean
        -rm $(NRMODS) $(MODS) $(SBR)

# Print updated .f* files
print:  print-stamp

print-stamp:    *.$(FC:77=)
        @-echo $?
        @-prl `match "$?" $(PGMS:.o=.$(FC:77=))`        \
                `nomatch "$?" $(PGMS:.o=.$(FC:77=)) | \
                grep -v '^f2ctmp_'` && touch $@ &

# Print everything
printall:
        @-rm print-stamp
        @-make print

# Prevent files from being erased by job interruption
.PRECIOUS:  $(SBR) $(PGMS) $(PGMS:.o=)
```

```
# Add to the suffix list
.SUFFIXES:  .f90 .M

# Compilation rules
.c.o:
        $(CC) $(CFLAGS) -c $<
.f90.o:
        $(FC) $(FFLAGS) -c $<

# Prepare module files
.f.M .f90.M:
        rmlib $(SBR) $(@:.M=)
        rmobj $(@:.M)
        $(FC) $(FFLAGS) -c $<
        @ar ruv $(SBR) $*.o
        @rm $*.o

# Library rules
.c.a:
        $(CC) $(CFLAGS) -c $<
        @ar ruv $@ $*.o
        @rm $*.o
.f.a .f90.a:
        $(FC) $(FFLAGS) -c $<
        @ar ruv $@ $*.o
        @rm $*.o
```

[Makefile-sun4]

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/26 21:13:25 ayahil>

!!! Log-likelihood function and its gradient wrt to sig & lambda for factor
!!! analysis with missing data. Called by the minimization routine FRPRMN.
!!! NOTE: In this routine V is the *reduced* covariance matrix.

FUNCTION Ml( &
  & lambda, &          ! Loading matrix
  & sig, &             ! Independent standard deviations
  & dlambda, &         ! derivative of function wrt lambda
  & dsig &             ! derivative of function wrt sigma
  & ) RESULT( out )

 USE Interfaces, ONLY: Covar
 USE Nrtype
 USE Nrutil, ONLY: Assert_eq, Diagadd, Get_diag, Outerprod
 USE Parm, ONLY: k, n, norm, p, r, sm, w
 USE Utils, ONLY: Dmatmul_l, Dmatmul_r, Spd
 IMPLICIT NONE
                    ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:) :: sig
 REAL(sp), INTENT(in), DIMENSION(:,:) :: lambda
 REAL(sp), OPTIONAL, INTENT(out), DIMENSION(p) :: dsig
 REAL(sp), OPTIONAL, INTENT(out), DIMENSION(p,k) :: dlambda
 REAL(sp) :: out
                    ! Locals
 INTEGER :: chk, i, m
 INTEGER, POINTER, DIMENSION(:) :: idx
 REAL(sp) :: dnorm, ldet
 REAL(sp), ALLOCATABLE, DIMENSION(:) :: b, x
 REAL(sp), ALLOCATABLE, DIMENSION(:,:) :: vinv
 REAL(sp), DIMENSION(p,p) :: v
                    ! Check sizes
 chk = Assert_eq( k, SIZE(lambda,2), ' Ml-k' )
 chk = Assert_eq( n, SIZE(r,1), SIZE(sm), SIZE(w), ' Ml-n' )
 chk = Assert_eq( (/ p, SIZE(lambda,1), SIZE(r,2), SIZE(sig), SIZE(v,1), &
    & SIZE(v,2) /), ' Ml-p' )
 IF( PRESENT(dsig) ) THEN
   chk = Assert_eq( p, SIZE(dsig), ' Ml-p' )
 END IF
 IF( PRESENT( dlambda) ) THEN
```

```
  chk = Assert_eq( k, SIZE(dlambda,2), ' Ml-k' )
  chk = Assert_eq( p, SIZE(dlambda,1), ' Ml-p' )
END IF
                    ! Initialization
norm = 0.0_sp
out = 0.0_sp
IF( PRESENT(dsig) ) THEN
  dsig = 0.0_sp
END IF
IF( PRESENT(dlambda) ) THEN
  dlambda = 0.0_sp
END IF
v = MATMUL( lambda, TRANSPOSE(lambda) )
CALL Diagadd( v, 1.0_sp )
                    ! Log-likelihood function
Data_loop: DO i = 1,n
  idx => sm(i)%idx
  m = SIZE(idx)
  IF( m > 0 ) THEN
    ALLOCATE( b(m), x(m) )
    b = r(i,idx)/sig(idx)
    IF( PRESENT(dsig) .OR. PRESENT(dlambda) ) THEN
      ALLOCATE( vinv(m,m) )
      CALL Spd( v(idx,idx), ainv=vinv, b=b, ldet=ldet, x=x )
      IF( PRESENT(dlambda) ) THEN
        dlambda(idx,:) = dlambda(idx,:) &
            & + w(i)*MATMUL( vinv - Outerprod( x, x ), lambda(idx,:) )
      END IF
      IF( PRESENT(dsig) ) THEN
        dsig(idx) = dsig(idx) + w(i)*(1.0_sp - b*x)
      END IF
      DEALLOCATE( vinv )
    ELSE
      CALL Spd( v(idx,idx), b=b, ldet=ldet, x=x )
    END IF
    dnorm = DOT_PRODUCT( b, x )
    DEALLOCATE( b, x )
    norm = norm + w(i)*dnorm
    out = out + w(i)*(ldet + dnorm + SUM( LOG( sig(idx)**2 ) ))
  END IF
END DO Data_loop
                    ! Optional gradient of likelihood function,
                    ! including penalty function
IF( PRESENT(dsig) ) THEN
  dsig = 2.0_sp*dsig/sig
END IF
```

```
  IF( PRESENT(dlambda) ) THEN
    dlambda = 2.0_sp*dlambda
  END IF

END FUNCTION Ml
```

```
!*****************************************
! Copyright (C) 1997 by Amos Yahil.
! All Rights Reserved.
! Based on (C) Numerical Recipes software.
!*****************************************
! Time-stamp: <97/03/28 13:04:47 ayahil>
SUBROUTINE mnbrak(ax,bx,cx,fa,fb,fc,func,lox,hix)
 USE nrtype; USE nrutil, ONLY : nrerror,swap
 IMPLICIT NONE
 REAL(SP), INTENT(INOUT) :: ax,bx
 REAL(SP), INTENT(OUT) :: cx,fa,fb,fc
 REAL(SP), OPTIONAL, INTENT(IN) :: hix,lox
 INTERFACE
   FUNCTION func(x,dx)
    USE nrtype
    IMPLICIT NONE
    REAL(SP), INTENT(IN) :: x
    REAL(SP), OPTIONAL, INTENT(OUT) :: dx
    REAL(SP) :: func
   END FUNCTION func
 END INTERFACE
 REAL(SP), PARAMETER :: GOLD=1.618034_sp,GLIMIT=100.0_sp,TINY=1.0e-20_sp
 REAL(SP) :: fu,q,r,u,ulim
 IF (PRESENT(lox)) THEN
   IF (MIN(ax,bx) < lox) CALL Nrerror( 'Mnbrak: min(ax,bx) < lox' )
 END IF
 IF (PRESENT(hix)) THEN
   IF (MAX(ax,bx) > hix) CALL Nrerror( 'Mnbrak: max(ax,bx) > hix' )
 END IF
 fa=func(ax)
 fb=func(bx)
 IF (fb > fa) THEN
   CALL swap(ax,bx)
   CALL swap(fa,fb)
 END IF
 cx=bx+GOLD*(bx-ax)
 IF (PRESENT(lox)) THEN
   IF (cx <= lox) THEN
     cx=lox
     RETURN
   END IF
 END IF
 IF (PRESENT(hix)) THEN
   IF (cx >= hix) THEN
     cx=hix
     RETURN
```

```
    END IF
  END IF
  fc=func(cx)
  DO
    IF (fb < fc) RETURN
    IF (PRESENT(lox)) THEN
      IF (cx == lox) RETURN
    END IF
    IF (PRESENT(hix)) THEN
      IF (cx == hix) RETURN
    END IF
    r=(bx-ax)*(fb-fc)
    q=(bx-cx)*(fb-fa)
    u=bx-((bx-cx)*q-(bx-ax)*r)/(2.0_sp*SIGN(MAX(ABS(q-r),TINY),q-r))
    IF (PRESENT(lox)) u=MAX(u,lox)
    IF (PRESENT(hix)) u=MIN(u,hix)
    ulim=bx+GLIMIT*(cx-bx)
    IF (PRESENT(lox)) ulim=MAX(ulim,lox)
    IF (PRESENT(hix)) ulim=MIN(ulim,hix)
    IF ((bx-u)*(u-cx) > 0.0) THEN
      fu=func(u)
      IF (fu < fc) THEN
        ax=bx
        fa=fb
        bx=u
        fb=fu
        RETURN
      ELSE IF (fu > fb) THEN
        cx=u
        fc=fu
        RETURN
      END IF
      u=cx+GOLD*(cx-bx)
      IF (PRESENT(lox)) u=MAX(u,lox)
      IF (PRESENT(hix)) u=MIN(u,hix)
      fu=func(u)
    ELSE IF ((cx-u)*(u-ulim) > 0.0) THEN
      fu=func(u)
      IF (fu < fc) THEN
        bx=cx
        cx=u
        u=cx+GOLD*(cx-bx)
        IF (PRESENT(lox)) u=MAX(u,lox)
        IF (PRESENT(hix)) u=MIN(u,hix)
        CALL shft(fb,fc,fu,func(u))
      END IF
```

```
    ELSE IF ((u-ulim)*(ulim-cx) >= 0.0) THEN
      u=ulim
      fu=func(u)
    ELSE
      u=cx+GOLD*(cx-bx)
      IF (PRESENT(lox)) u=MAX(u,lox)
      IF (PRESENT(hix)) u=MIN(u,hix)
      fu=func(u)
    END IF
    CALL shft(ax,bx,cx,u)
    CALL shft(fa,fb,fc,fu)
  END DO
CONTAINS
                        !BL
  SUBROUTINE shft(a,b,c,d)
    REAL(SP), INTENT(OUT) :: a
    REAL(SP), INTENT(INOUT) :: b,c
    REAL(SP), INTENT(IN) :: d
    a=b
    b=c
    c=d
  END SUBROUTINE shft
END SUBROUTINE mnbrak
```

```
!*****************************************
! Copyright (C) 1997 by Amos Yahil.
! All Rights Reserved.
! Based on (C) Numerical Recipes software.
!*****************************************
! Time-stamp: <98/06/15 20:16:28 ayahil>
MODULE mynr

 INTERFACE
   FUNCTION asolve( r ) RESULT( out )
    USE nrtype
    IMPLICIT NONE
    REAL(DP), DIMENSION(:), INTENT(IN) :: r
    REAL(DP), DIMENSION(SIZE(r)) :: out
   END FUNCTION asolve
 END INTERFACE

 INTERFACE
   FUNCTION atimes( x ) RESULT( out )
    USE nrtype
    IMPLICIT NONE
    REAL(DP), DIMENSION(:), INTENT(IN) :: x
    REAL(DP), DIMENSION(SIZE(x)) :: out
   END FUNCTION atimes
 END INTERFACE

 INTERFACE bcucof
   SUBROUTINE bcucof_r(y,y1,y2,y12,d1,d2,c)
    USE nrtype
    IMPLICIT NONE
    REAL(SP), INTENT(IN) :: d1,d2
    REAL(SP), DIMENSION(4), INTENT(IN) :: y,y1,y2,y12
    REAL(SP), DIMENSION(4,4), INTENT(OUT) :: c
   END SUBROUTINE bcucof_r
!!!BL
   SUBROUTINE bcucof_c(y,y1,y2,y12,d1,d2,c)
    USE nrtype
    IMPLICIT NONE
    REAL(SP), INTENT(IN) :: d1,d2
    COMPLEX(SP), DIMENSION(4), INTENT(IN) :: y,y1,y2,y12
    COMPLEX(SP), DIMENSION(4,4), INTENT(OUT) :: c
   END SUBROUTINE bcucof_c
 END INTERFACE

 INTERFACE bcuint
   SUBROUTINE bcuint_r(y,y1,y2,y12,x1l,x1u,x2l,x2u,x1,x2,ansy,ansy1,ansy2)
```

```
      USE nrtype
      IMPLICIT NONE
      REAL(SP), DIMENSION(4), INTENT(IN) :: y,y1,y2,y12
      REAL(SP), INTENT(IN) :: x1l,x1u,x2l,x2u,x1,x2
      REAL(SP), INTENT(OUT) :: ansy,ansy1,ansy2
    END SUBROUTINE bcuint_r
!!!BL
    SUBROUTINE bcuint_c(y,y1,y2,y12,x1l,x1u,x2l,x2u,x1,x2,ansy,ansy1,ansy2)
      USE nrtype
      IMPLICIT NONE
      COMPLEX(SP), DIMENSION(4), INTENT(IN) :: y,y1,y2,y12
      REAL(SP), INTENT(IN) :: x1l,x1u,x2l,x2u,x1,x2
      COMPLEX(SP), INTENT(OUT) :: ansy,ansy1,ansy2
    END SUBROUTINE bcuint_c
  END INTERFACE

  INTERFACE
    FUNCTION brent(ax,bx,cx,func,tol,xmin)
      USE nrtype
      IMPLICIT NONE
      REAL(SP), INTENT(IN) :: ax,bx,cx,tol
      REAL(SP), INTENT(OUT) :: xmin
      REAL(SP) :: brent
      INTERFACE
        FUNCTION func(x,dx)
          USE nrtype
          IMPLICIT NONE
          REAL(SP), INTENT(IN) :: x
          REAL(SP), OPTIONAL, INTENT(OUT) :: dx
          REAL(SP) :: func
        END FUNCTION func
      END INTERFACE
    END FUNCTION brent
  END INTERFACE

  INTERFACE
    FUNCTION dbrent(ax,bx,cx,func,tol,xmin)
      USE nrtype
      IMPLICIT NONE
      REAL(SP), INTENT(IN) :: ax,bx,cx,tol
      REAL(SP), INTENT(OUT) :: xmin
      REAL(SP) :: dbrent
      INTERFACE
        FUNCTION func(x,dx)
          USE nrtype
          IMPLICIT NONE
```

```
      REAL(SP), INTENT(IN) :: x
      REAL(SP), OPTIONAL, INTENT(OUT) :: dx
      REAL(SP) :: func
     END FUNCTION func
    END INTERFACE
  END FUNCTION dbrent
END INTERFACE

INTERFACE
  SUBROUTINE dfrprmn(p,abstol,reltol,iter,fret,lop,hip,pr)
   USE nrtype
   IMPLICIT NONE
   INTEGER(I4B), INTENT(inout) :: iter
   LOGICAL, OPTIONAL, INTENT(in) :: pr
   REAL(SP), INTENT(in) :: abstol, reltol
   REAL(SP), INTENT(out) :: fret
   REAL(SP), DIMENSION(:), INTENT(inout) :: p
   REAL(SP), DIMENSION(:), OPTIONAL, INTENT(in) :: hip, lop
  END SUBROUTINE dfrprmn
END INTERFACE

INTERFACE
  SUBROUTINE dlinmin(p,xi,fret,dx,lop,hip)
   USE nrtype
   IMPLICIT NONE
   REAL(SP), INTENT(OUT) :: fret
   REAL(SP), DIMENSION(:), TARGET, INTENT(INOUT) :: p,xi
   REAL(SP), OPTIONAL :: dx
   REAL(SP), DIMENSION(:), OPTIONAL, INTENT(in) :: hip,lop
  END SUBROUTINE dlinmin
END INTERFACE

INTERFACE
  SUBROUTINE frprmn(p,abstol,reltol,iter,fret,lop,hip,ipr)
   USE nrtype
   IMPLICIT NONE
   INTEGER(I4B), INTENT(inout) :: iter
   INTEGER, OPTIONAL, INTENT(in) :: ipr
   REAL(SP), INTENT(in) :: abstol, reltol
   REAL(SP), INTENT(out) :: fret
   REAL(SP), DIMENSION(:), INTENT(inout) :: p
   REAL(SP), DIMENSION(:), OPTIONAL, INTENT(in) :: hip, lop
  END SUBROUTINE frprmn
END INTERFACE

INTERFACE
```

```
  SUBROUTINE lincg(b,x,tol,itmax)
   USE nrtype
   IMPLICIT NONE
   INTEGER, INTENT(IN) :: itmax
   REAL(DP), INTENT(IN) :: tol
   REAL(DP), DIMENSION(:), INTENT(IN) :: b
   REAL(DP), DIMENSION(SIZE(b)), INTENT(INOUT) :: x
  END SUBROUTINE lincg
END INTERFACE

INTERFACE
  SUBROUTINE linmin(p,xi,fret,dx,lop,hip)
   USE nrtype
   IMPLICIT NONE
   REAL(SP), INTENT(OUT) :: fret
   REAL(SP), DIMENSION(:), TARGET, INTENT(INOUT) :: p,xi
   REAL(SP), OPTIONAL :: dx
   REAL(SP), DIMENSION(:), OPTIONAL, INTENT(in) :: hip,lop
  END SUBROUTINE linmin
END INTERFACE

INTERFACE
  SUBROUTINE mnbrak(ax,bx,cx,fa,fb,fc,func,lox,hix)
   USE nrtype
   IMPLICIT NONE
   REAL(SP), INTENT(INOUT) :: ax,bx
   REAL(SP), INTENT(OUT) :: cx,fa,fb,fc
   REAL(SP), OPTIONAL, INTENT(IN) :: hix,lox
   INTERFACE
     FUNCTION func(x,dx)
      USE nrtype
      IMPLICIT NONE
      REAL(SP), INTENT(IN) :: x
      REAL(SP), OPTIONAL, INTENT(OUT) :: dx
      REAL(SP) :: func
     END FUNCTION func
   END INTERFACE
  END SUBROUTINE mnbrak
END INTERFACE

INTERFACE my_ran1
  SUBROUTINE my_ran1_s(harvest)
   USE nrtype
   REAL(SP), INTENT(OUT) :: harvest
  END SUBROUTINE my_ran1_s
```

```
    SUBROUTINE my_ran1_v(harvest)
      USE nrtype
      REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
    END SUBROUTINE my_ran1_v
  END INTERFACE

END MODULE mynr
```

```
SUBROUTINE my_ran1_s(harvest)
USE nrtype
USE my_ran_state, ONLY: K4B,amm,lenran,my_ran_init, &
        iran0,jran0,kran0,nran0,mran0,rans
IMPLICIT NONE
REAL(SP), INTENT(OUT) :: harvest
IF (lenran < 1) CALL my_ran_init(1)
rans=iran0-kran0
IF (rans < 0) rans=rans+2147483579_k4b
iran0=jran0
jran0=kran0
kran0=rans
nran0=IEOR(nran0,ISHFT(nran0,13))
nran0=IEOR(nran0,ISHFT(nran0,-17))
nran0=IEOR(nran0,ISHFT(nran0,5))
IF (nran0 == 1) nran0=270369_k4b
mran0=IEOR(mran0,ISHFT(mran0,5))
mran0=IEOR(mran0,ISHFT(mran0,-13))
mran0=IEOR(mran0,ISHFT(mran0,6))
rans=IEOR(nran0,rans)+mran0
harvest=amm*MERGE(rans,NOT(rans), rans<0 )
END SUBROUTINE my_ran1_s

SUBROUTINE my_ran1_v(harvest)
USE nrtype
USE my_ran_state, ONLY: K4B,amm,lenran,my_ran_init, &
        iran,jran,kran,nran,mran,ranv
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
INTEGER(K4B) :: n
n=SIZE(harvest)
IF (lenran < n+1) CALL my_ran_init(n+1)
ranv(1:n)=iran(1:n)-kran(1:n)
WHERE (ranv(1:n) < 0) ranv(1:n)=ranv(1:n)+2147483579_k4b
iran(1:n)=jran(1:n)
jran(1:n)=kran(1:n)
kran(1:n)=ranv(1:n)
nran(1:n)=IEOR(nran(1:n),ISHFT(nran(1:n),13))
nran(1:n)=IEOR(nran(1:n),ISHFT(nran(1:n),-17))
nran(1:n)=IEOR(nran(1:n),ISHFT(nran(1:n),5))
WHERE (nran(1:n) == 1) nran(1:n)=270369_k4b
mran(1:n)=IEOR(mran(1:n),ISHFT(mran(1:n),5))
mran(1:n)=IEOR(mran(1:n),ISHFT(mran(1:n),-13))
mran(1:n)=IEOR(mran(1:n),ISHFT(mran(1:n),6))
ranv(1:n)=IEOR(nran(1:n),ranv(1:n))+mran(1:n)
harvest=amm*MERGE(ranv(1:n),NOT(ranv(1:n)), ranv(1:n)<0 )
```

END SUBROUTINE my_ran1_v

```
MODULE my_ran_state
	USE nrtype
	IMPLICIT NONE
	INTEGER, PARAMETER :: K4B=SELECTED_INT_KIND(9)
	INTEGER(K4B), PARAMETER :: hg=HUGE(1_K4B), hgm=-hg, hgng=hgm-1
	INTEGER(K4B), SAVE :: lenran=0, seq=0
	INTEGER(K4B), SAVE :: iran0,jran0,kran0,nran0,mran0,rans
	INTEGER(K4B), DIMENSION(:,:), POINTER, SAVE :: ranseeds
	INTEGER(K4B), DIMENSION(:), POINTER, SAVE :: iran,jran,kran, &
		nran,mran,ranv
	REAL(SP), SAVE :: amm
	INTERFACE my_ran_hash
		MODULE PROCEDURE my_ran_hash_s, my_ran_hash_v
	END INTERFACE
CONTAINS
!BL
	SUBROUTINE my_ran_init(length)
	USE nrtype; USE nrutil, ONLY : arth,nrerror,reallocate
	IMPLICIT NONE
	INTEGER(K4B), INTENT(IN) :: length
	INTEGER(K4B) :: NEW,j,hgt
	IF (length < lenran) RETURN
	hgt=hg
	IF (hg /= 2147483647) CALL nrerror('my_ran_init: arith assump 1 fails')
	IF (hgng >= 0)       CALL nrerror('my_ran_init: arith assump 2 fails')
	IF (hgt+1 /= hgng)   CALL nrerror('my_ran_init: arith assump 3 fails')
	IF (NOT(hg) >= 0)    CALL nrerror('my_ran_init: arith assump 4 fails')
	IF (NOT(hgng) < 0)   CALL nrerror('my_ran_init: arith assump 5 fails')
	IF (hg+hgng >= 0)    CALL nrerror('my_ran_init: arith assump 6 fails')
	IF (NOT(-1_k4b) < 0) CALL nrerror('my_ran_init: arith assump 7 fails')
	IF (NOT(0_k4b) >= 0) CALL nrerror('my_ran_init: arith assump 8 fails')
	IF (NOT(1_k4b) >= 0) CALL nrerror('my_ran_init: arith assump 9 fails')
	IF (lenran > 0) THEN
		ranseeds=>reallocate(ranseeds,length,5)
		ranv=>reallocate(ranv,length-1)
		NEW=lenran+1
	ELSE
		ALLOCATE(ranseeds(length,5))
		ALLOCATE(ranv(length-1))
		NEW=1
		amm=NEAREST(1.0_sp,-1.0_sp)/hgng
		IF (amm*hgng >= 1.0 .OR. amm*hgng <= 0.0) &
			CALL nrerror('my_ran_init: arth assump 10 fails')
	END IF
	ranseeds(NEW:,1)=seq
	ranseeds(NEW:,2:5)=SPREAD(arth(NEW,1,SIZE(ranseeds(NEW:,1))),2,4)
```

[my_ran_state.f90]

```
            iran0=ranseeds(1,1)
            jran0=ranseeds(1,2)
            kran0=ranseeds(1,3)
            mran0=ranseeds(1,4)
            nran0=ranseeds(1,5)
        ELSE IF (PRESENT(get)) THEN
            IF (lenran == 0) RETURN
            ranseeds(1,1:5)=(/ iran0,jran0,kran0,mran0,nran0 /)
            get=RESHAPE(ranseeds,SHAPE(get))
        ELSE IF (PRESENT(SEQUENCE)) THEN
            CALL my_ran_deallocate
            seq=SEQUENCE
        END IF
        END SUBROUTINE my_ran_seed
!BL
        SUBROUTINE my_ran_hash_s(il,ir)
        IMPLICIT NONE
        INTEGER(K4B), INTENT(INOUT) :: il,ir
        INTEGER(K4B) :: is,j
        DO j=1,4
            is=ir
            ir=IEOR(ir,ISHFT(ir,5))+1422217823
            ir=IEOR(ir,ISHFT(ir,-16))+1842055030
            ir=IEOR(ir,ISHFT(ir,9))+80567781
            ir=IEOR(il,ir)
            il=is
        END DO
        END SUBROUTINE my_ran_hash_s
!BL
        SUBROUTINE my_ran_hash_v(il,ir)
        IMPLICIT NONE
        INTEGER(K4B), DIMENSION(:), INTENT(INOUT) :: il,ir
        INTEGER(K4B), DIMENSION(SIZE(il)) :: is
        INTEGER(K4B) :: j
        DO j=1,4
            is=ir
            ir=IEOR(ir,ISHFT(ir,5))+1422217823
            ir=IEOR(ir,ISHFT(ir,-16))+1842055030
            ir=IEOR(ir,ISHFT(ir,9))+80567781
            ir=IEOR(il,ir)
            il=is
        END DO
        END SUBROUTINE my_ran_hash_v
END MODULE my_ran_state
```

```
        DO j=1,4
                CALL my_ran_hash(ranseeds(NEW:,j),ranseeds(NEW:,j+1))
        END DO
        WHERE (ranseeds(NEW:,1:3) < 0) &
                ranseeds(NEW:,1:3)=NOT(ranseeds(NEW:,1:3))
        WHERE (ranseeds(NEW:,4:5) == 0) ranseeds(NEW:,4:5)=1
        IF (NEW == 1) THEN
                iran0=ranseeds(1,1)
                jran0=ranseeds(1,2)
                kran0=ranseeds(1,3)
                mran0=ranseeds(1,4)
                nran0=ranseeds(1,5)
                rans=nran0
        END IF
        IF (length > 1) THEN
                iran => ranseeds(2:,1)
                jran => ranseeds(2:,2)
                kran => ranseeds(2:,3)
                mran => ranseeds(2:,4)
                nran => ranseeds(2:,5)
                ranv = nran
        END IF
        lenran=length
        END SUBROUTINE my_ran_init
!BL
        SUBROUTINE my_ran_deallocate
        IF (lenran > 0) THEN
                DEALLOCATE(ranseeds,ranv)
                NULLIFY(ranseeds,ranv,iran,jran,kran,mran,nran)
                lenran = 0
        END IF
        END SUBROUTINE my_ran_deallocate
!BL
        SUBROUTINE my_ran_seed(SEQUENCE,size,put,get)
        IMPLICIT NONE
        INTEGER, OPTIONAL, INTENT(IN) :: SEQUENCE
        INTEGER, OPTIONAL, INTENT(OUT) :: size
        INTEGER, DIMENSION(:), OPTIONAL, INTENT(IN) :: put
        INTEGER, DIMENSION(:), OPTIONAL, INTENT(OUT) :: get
        IF (PRESENT(size)) THEN
                size=5*lenran
        ELSE IF (PRESENT(put)) THEN
                IF (lenran == 0) RETURN
                ranseeds=RESHAPE(put,SHAPE(ranseeds))
                WHERE (ranseeds(:,1:3) < 0) ranseeds(:,1:3)=NOT(ranseeds(:,1:3))
                WHERE (ranseeds(:,4:5) == 0) ranseeds(:,4:5)=1
```

```
!*****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!*****************************************
! Time-stamp: <98/06/17 21:45:45 ayahil>

!!! Get a new search direction from a saddle point by finding the direction
!!! with the smallest negative curvature (biggest in absolute value).
!!! Note, in this routine V is the *reduced* covariance matrix.

FUNCTION Next( &
  & z &                 ! Minimization variables
  & ) RESULT( out )

 USE Nr, ONLY: Eigsrt, Tqli, Tred2
 USE Nrtype
 USE Nrutil, ONLY: Assert_eq, Diagadd, Outerprod
 USE Parm, ONLY: k, n, p, r, sm
 USE Utils, ONLY: Spd
 IMPLICIT NONE
                       ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:) :: z
 REAL(sp), DIMENSION(p) :: out
                       ! Locals
 INTEGER :: chk, i, m
 INTEGER, DIMENSION(:), POINTER :: idx
 REAL(sp), ALLOCATABLE, DIMENSION(:) :: x
 REAL(sp), ALLOCATABLE, DIMENSION(:,:) :: vinv
 REAL(sp), DIMENSION(p) :: d, e, sig
 REAL(sp), DIMENSION(p,k) :: lambda
 REAL(sp), DIMENSION(p,p) :: omega, v
                       ! Check sizes
 chk = Assert_eq( p*(k+1), SIZE(z), ' Next-z' )
                       ! Derived sizes
 sig = z(:p)
 lambda = RESHAPE( z(p+1:), SHAPE(lambda) )
                       ! Check zeroing of last factor
 IF( ANY( lambda(:,k) /= 0.0_sp ) ) &
    & STOP 'Added parameters must be initialized to zero'
                       ! Reduced covariance matrix
 v = MATMUL( lambda, TRANSPOSE(lambda) )
 CALL Diagadd( v, 1.0_sp )
                       ! Second-derivative matrix (minus half of)
 omega = 0.0
 Data_loop: DO i = 1,n
```

```
    idx => sm(i)%idx
    m = SIZE(idx)
    IF( m > 0 ) THEN
      ALLOCATE( vinv(m,m), x(m) )
      CALL Spd( v(idx,idx), ainv=vinv, b=r(i,idx)/sig(idx), x=x )
      omega(idx,idx) = omega(idx,idx) + Outerprod( x, x ) - vinv
      DEALLOCATE( vinv, x )
    END IF
  END DO Data_loop
                    ! Find maximum eigenvalue/vector
  CALL Tred2( omega, d, e )
  CALL Tqli( d, e, omega )
  CALL Eigsrt( d, omega )
                    ! Maximum eigenvector, if eigenvalue > 0
  IF( d(1) > SQRT( EPSILON( 1.0_sp ) ) ) THEN
    out = omega(:,1)/SQRT( d(1) )
  ELSE
    out = 0.0
  END IF

END FUNCTION Next
```

```
MODULE nr
	INTERFACE
		SUBROUTINE airy(x,ai,bi,aip,bip)
		USE nrtype
		REAL(SP), INTENT(IN) :: x
		REAL(SP), INTENT(OUT) :: ai,bi,aip,bip
		END SUBROUTINE airy
	END INTERFACE
	INTERFACE
		SUBROUTINE amebsa(p,y,pb,yb,ftol,func,iter,temptr)
		USE nrtype
		INTEGER(I4B), INTENT(INOUT) :: iter
		REAL(SP), INTENT(INOUT) :: yb
		REAL(SP), INTENT(IN) :: ftol,temptr
		REAL(SP), DIMENSION(:), INTENT(INOUT) :: y,pb
		REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: p
		INTERFACE
			FUNCTION func(x)
			USE nrtype
			REAL(SP), DIMENSION(:), INTENT(IN) :: x
			REAL(SP) :: func
			END FUNCTION func
		END INTERFACE
		END SUBROUTINE amebsa
	END INTERFACE
	INTERFACE
		SUBROUTINE amoeba(p,y,ftol,func,iter)
		USE nrtype
		INTEGER(I4B), INTENT(OUT) :: iter
		REAL(SP), INTENT(IN) :: ftol
		REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
		REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: p
		INTERFACE
			FUNCTION func(x)
			USE nrtype
			REAL(SP), DIMENSION(:), INTENT(IN) :: x
			REAL(SP) :: func
			END FUNCTION func
		END INTERFACE
		END SUBROUTINE amoeba
	END INTERFACE
	INTERFACE
		SUBROUTINE anneal(x,y,iorder)
		USE nrtype
		INTEGER(I4B), DIMENSION(:), INTENT(INOUT) :: iorder
		REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
```

```
        END SUBROUTINE anneal
    END INTERFACE
    INTERFACE
        SUBROUTINE asolve(b,x,itrnsp)
        USE nrtype
        REAL(DP), DIMENSION(:), INTENT(IN) :: b
        REAL(DP), DIMENSION(:), INTENT(OUT) :: x
        INTEGER(I4B), INTENT(IN) :: itrnsp
        END SUBROUTINE asolve
    END INTERFACE
    INTERFACE
        SUBROUTINE atimes(x,r,itrnsp)
        USE nrtype
        REAL(DP), DIMENSION(:), INTENT(IN) :: x
        REAL(DP), DIMENSION(:), INTENT(OUT) :: r
        INTEGER(I4B), INTENT(IN) :: itrnsp
        END SUBROUTINE atimes
    END INTERFACE
    INTERFACE
        SUBROUTINE avevar(data,ave,var)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: data
        REAL(SP), INTENT(OUT) :: ave,var
        END SUBROUTINE avevar
    END INTERFACE
    INTERFACE
        SUBROUTINE balanc(a)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
        END SUBROUTINE balanc
    END INTERFACE
    INTERFACE
        SUBROUTINE banbks(a,m1,m2,al,indx,b)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: m1,m2
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: indx
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: a,al
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: b
        END SUBROUTINE banbks
    END INTERFACE
    INTERFACE
        SUBROUTINE bandec(a,m1,m2,al,indx,d)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: m1,m2
        INTEGER(I4B), DIMENSION(:), INTENT(OUT) :: indx
        REAL(SP), INTENT(OUT) :: d
```

```
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: al
        END SUBROUTINE bandec
    END INTERFACE
    INTERFACE
        SUBROUTINE banmul(a,m1,m2,x,b)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: m1,m2
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
        REAL(SP), DIMENSION(:), INTENT(OUT) :: b
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: a
        END SUBROUTINE banmul
    END INTERFACE
    INTERFACE
        SUBROUTINE bcucof(y,y1,y2,y12,d1,d2,c)
        USE nrtype
        REAL(SP), INTENT(IN) :: d1,d2
        REAL(SP), DIMENSION(4), INTENT(IN) :: y,y1,y2,y12
        REAL(SP), DIMENSION(4,4), INTENT(OUT) :: c
        END SUBROUTINE bcucof
    END INTERFACE
    INTERFACE
        SUBROUTINE bcuint(y,y1,y2,y12,x1l,x1u,x2l,x2u,x1,x2,ansy,&
            ansy1,ansy2)
        USE nrtype
        REAL(SP), DIMENSION(4), INTENT(IN) :: y,y1,y2,y12
        REAL(SP), INTENT(IN) :: x1l,x1u,x2l,x2u,x1,x2
        REAL(SP), INTENT(OUT) :: ansy,ansy1,ansy2
        END SUBROUTINE bcuint
    END INTERFACE
    INTERFACE beschb
        SUBROUTINE beschb_s(x,gam1,gam2,gampl,gammi)
        USE nrtype
        REAL(DP), INTENT(IN) :: x
        REAL(DP), INTENT(OUT) :: gam1,gam2,gampl,gammi
        END SUBROUTINE beschb_s
!BL
        SUBROUTINE beschb_v(x,gam1,gam2,gampl,gammi)
        USE nrtype
        REAL(DP), DIMENSION(:), INTENT(IN) :: x
        REAL(DP), DIMENSION(:), INTENT(OUT) :: gam1,gam2,gampl,gammi
        END SUBROUTINE beschb_v
    END INTERFACE
    INTERFACE bessi
        FUNCTION bessi_s(n,x)
        USE nrtype
```

```
            INTEGER(I4B), INTENT(IN) :: n
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: bessi_s
            END FUNCTION bessi_s
!BL
            FUNCTION bessi_v(n,x)
            USE nrtype
            INTEGER(I4B), INTENT(IN) :: n
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: bessi_v
            END FUNCTION bessi_v
        END INTERFACE
        INTERFACE bessi0
            FUNCTION bessi0_s(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: bessi0_s
            END FUNCTION bessi0_s
!BL
            FUNCTION bessi0_v(x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: bessi0_v
            END FUNCTION bessi0_v
        END INTERFACE
        INTERFACE bessi1
            FUNCTION bessi1_s(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: bessi1_s
            END FUNCTION bessi1_s
!BL
            FUNCTION bessi1_v(x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: bessi1_v
            END FUNCTION bessi1_v
        END INTERFACE
        INTERFACE
            SUBROUTINE bessik(x,xnu,ri,rk,rip,rkp)
            USE nrtype
            REAL(SP), INTENT(IN) :: x,xnu
            REAL(SP), INTENT(OUT) :: ri,rk,rip,rkp
            END SUBROUTINE bessik
        END INTERFACE
        INTERFACE bessj
```

```
            FUNCTION bessj_s(n,x)
            USE nrtype
            INTEGER(I4B), INTENT(IN) :: n
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: bessj_s
            END FUNCTION bessj_s
!BL
            FUNCTION bessj_v(n,x)
            USE nrtype
            INTEGER(I4B), INTENT(IN) :: n
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: bessj_v
            END FUNCTION bessj_v
        END INTERFACE
        INTERFACE bessj0
            FUNCTION bessj0_s(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: bessj0_s
            END FUNCTION bessj0_s
!BL
            FUNCTION bessj0_v(x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: bessj0_v
            END FUNCTION bessj0_v
        END INTERFACE
        INTERFACE bessj1
            FUNCTION bessj1_s(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: bessj1_s
            END FUNCTION bessj1_s
!BL
            FUNCTION bessj1_v(x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: bessj1_v
            END FUNCTION bessj1_v
        END INTERFACE
        INTERFACE bessjy
            SUBROUTINE bessjy_s(x,xnu,rj,ry,rjp,ryp)
            USE nrtype
            REAL(SP), INTENT(IN) :: x,xnu
            REAL(SP), INTENT(OUT) :: rj,ry,rjp,ryp
            END SUBROUTINE bessjy_s
```

```
!BL
        SUBROUTINE bessjy_v(x,xnu,rj,ry,rjp,ryp)
        USE nrtype
        REAL(SP), INTENT(IN) :: xnu
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
        REAL(SP), DIMENSION(:), INTENT(OUT) :: rj,rjp,ry,ryp
        END SUBROUTINE bessjy_v
    END INTERFACE
    INTERFACE bessk
        FUNCTION bessk_s(n,x)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP), INTENT(IN) :: x
        REAL(SP) :: bessk_s
        END FUNCTION bessk_s
!BL
        FUNCTION bessk_v(n,x)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
        REAL(SP), DIMENSION(size(x)) :: bessk_v
        END FUNCTION bessk_v
    END INTERFACE
    INTERFACE bessk0
        FUNCTION bessk0_s(x)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        REAL(SP) :: bessk0_s
        END FUNCTION bessk0_s
!BL
        FUNCTION bessk0_v(x)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
        REAL(SP), DIMENSION(size(x)) :: bessk0_v
        END FUNCTION bessk0_v
    END INTERFACE
    INTERFACE bessk1
        FUNCTION bessk1_s(x)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        REAL(SP) :: bessk1_s
        END FUNCTION bessk1_s
!BL
        FUNCTION bessk1_v(x)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
```

```
        REAL(SP), DIMENSION(size(x)) :: bessk1_v
        END FUNCTION bessk1_v
    END INTERFACE
    INTERFACE bessy
        FUNCTION bessy_s(n,x)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP), INTENT(IN) :: x
        REAL(SP) :: bessy_s
        END FUNCTION bessy_s
!BL
        FUNCTION bessy_v(n,x)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
        REAL(SP), DIMENSION(size(x)) :: bessy_v
        END FUNCTION bessy_v
    END INTERFACE
    INTERFACE bessy0
        FUNCTION bessy0_s(x)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        REAL(SP) :: bessy0_s
        END FUNCTION bessy0_s
!BL
        FUNCTION bessy0_v(x)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
        REAL(SP), DIMENSION(size(x)) :: bessy0_v
        END FUNCTION bessy0_v
    END INTERFACE
    INTERFACE bessy1
        FUNCTION bessy1_s(x)
        USE nrtype
        REAL(SP), INTENT(IN) ::.x
        REAL(SP) :: bessy1_s
        END FUNCTION bessy1_s
!BL
        FUNCTION bessy1_v(x)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
        REAL(SP), DIMENSION(size(x)) :: bessy1_v
        END FUNCTION bessy1_v
    END INTERFACE
    INTERFACE beta
        FUNCTION beta_s(z,w)
```

```
            USE nrtype
            REAL(SP), INTENT(IN) :: z,w
            REAL(SP) :: beta_s
            END FUNCTION beta_s
!BL
            FUNCTION beta_v(z,w)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: z,w
            REAL(SP), DIMENSION(size(z)) :: beta_v
            END FUNCTION beta_v
        END INTERFACE
        INTERFACE betacf
            FUNCTION betacf_s(a,b,x)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,b,x
            REAL(SP) :: betacf_s
            END FUNCTION betacf_s
!BL
            FUNCTION betacf_v(a,b,x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: a,b,x
            REAL(SP), DIMENSION(size(x)) :: betacf_v
            END FUNCTION betacf_v
        END INTERFACE
        INTERFACE betai
            FUNCTION betai_s(a,b,x)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,b,x
            REAL(SP) :: betai_s
            END FUNCTION betai_s
!BL
            FUNCTION betai_v(a,b,x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: a,b,x
            REAL(SP), DIMENSION(size(a)) :: betai_v
            END FUNCTION betai_v
        END INTERFACE
        INTERFACE bico
            FUNCTION bico_s(n,k)
            USE nrtype
            INTEGER(I4B), INTENT(IN) :: n,k
            REAL(SP) :: bico_s
            END FUNCTION bico_s
!BL
            FUNCTION bico_v(n,k)
            USE nrtype
```

```
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: n,k
        REAL(SP), DIMENSION(size(n)) :: bico_v
        END FUNCTION bico_v
END INTERFACE
INTERFACE
        FUNCTION bnldev(pp,n)
        USE nrtype
        REAL(SP), INTENT(IN) :: pp
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP) :: bnldev
        END FUNCTION bnldev
END INTERFACE
INTERFACE
        FUNCTION brent(ax,bx,cx,func,tol,xmin)
        USE nrtype
        REAL(SP), INTENT(IN) :: ax,bx,cx,tol
        REAL(SP), INTENT(OUT) :: xmin
        REAL(SP) :: brent
        INTERFACE
                FUNCTION func(x)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: func
                END FUNCTION func
        END INTERFACE
        END FUNCTION brent
END INTERFACE
INTERFACE
        SUBROUTINE broydn(x,check)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: x
        LOGICAL(LGT), INTENT(OUT) :: check
        END SUBROUTINE broydn
END INTERFACE
INTERFACE
        SUBROUTINE bsstep(y,dydx,x,htry,eps,yscal,hdid,hnext,derivs)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
        REAL(SP), DIMENSION(:), INTENT(IN) :: dydx,yscal
        REAL(SP), INTENT(INOUT) :: x
        REAL(SP), INTENT(IN) :: htry,eps
        REAL(SP), INTENT(OUT) :: hdid,hnext
        INTERFACE
                SUBROUTINE derivs(x,y,dydx)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
```

```
                REAL(SP), DIMENSION(:), INTENT(IN) :: y
                REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
                END SUBROUTINE derivs
            END INTERFACE
            END SUBROUTINE bsstep
        END INTERFACE
        INTERFACE
            SUBROUTINE caldat(julian,mm,id,iyyy)
            USE nrtype
            INTEGER(I4B), INTENT(IN) :: julian
            INTEGER(I4B), INTENT(OUT) :: mm,id,iyyy
            END SUBROUTINE caldat
        END INTERFACE
        INTERFACE
            FUNCTION chder(a,b,c)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,b
            REAL(SP), DIMENSION(:), INTENT(IN) :: c
            REAL(SP), DIMENSION(size(c)) :: chder
            END FUNCTION chder
        END INTERFACE
        INTERFACE chebev
            FUNCTION chebev_s(a,b,c,x)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,b,x
            REAL(SP), DIMENSION(:), INTENT(IN) :: c
            REAL(SP) :: chebev_s
            END FUNCTION chebev_s
!BL
            FUNCTION chebev_v(a,b,c,x)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,b
            REAL(SP), DIMENSION(:), INTENT(IN) :: c,x
            REAL(SP), DIMENSION(size(x)) :: chebev_v
            END FUNCTION chebev_v
        END INTERFACE
        INTERFACE
            FUNCTION chebft(a,b,n,func)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,b
            INTEGER(I4B), INTENT(IN) :: n
            REAL(SP), DIMENSION(n) :: chebft
            INTERFACE
                FUNCTION func(x)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
```

```
            REAL(SP), DIMENSION(size(x)) :: func
            END FUNCTION func
        END INTERFACE
        END FUNCTION chebft
END INTERFACE
INTERFACE
        FUNCTION chebpc(c)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: c
        REAL(SP), DIMENSION(size(c)) :: chebpc
        END FUNCTION chebpc
END INTERFACE
INTERFACE
        FUNCTION chint(a,b,c)
        USE nrtype
        REAL(SP), INTENT(IN) :: a,b
        REAL(SP), DIMENSION(:), INTENT(IN) :: c
        REAL(SP), DIMENSION(size(c)) :: chint
        END FUNCTION chint
END INTERFACE
INTERFACE
        SUBROUTINE choldc(a,p)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
        REAL(SP), DIMENSION(:), INTENT(OUT) :: p
        END SUBROUTINE choldc
END INTERFACE
INTERFACE
        SUBROUTINE cholsl(a,p,b,x)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: a
        REAL(SP), DIMENSION(:), INTENT(IN) :: p,b
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: x
        END SUBROUTINE cholsl
END INTERFACE
INTERFACE
        SUBROUTINE chsone(bins,ebins,knstrn,df,chsq,prob)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: knstrn
        REAL(SP), INTENT(OUT) :: df,chsq,prob
        REAL(SP), DIMENSION(:), INTENT(IN) :: bins,ebins
        END SUBROUTINE chsone
END INTERFACE
INTERFACE
        SUBROUTINE chstwo(bins1,bins2,knstrn,df,chsq,prob)
        USE nrtype
```

```
        INTEGER(I4B), INTENT(IN) :: knstrn
        REAL(SP), INTENT(OUT) :: df,chsq,prob
        REAL(SP), DIMENSION(:), INTENT(IN) :: bins1,bins2
        END SUBROUTINE chstwo
    END INTERFACE
    INTERFACE
        SUBROUTINE cisi(x,ci,si)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        REAL(SP), INTENT(OUT) :: ci,si
        END SUBROUTINE cisi
    END INTERFACE
    INTERFACE
        SUBROUTINE cntab1(nn,chisq,df,prob,cramrv,ccc)
        USE nrtype
        INTEGER(I4B), DIMENSION(:,:), INTENT(IN) :: nn
        REAL(SP), INTENT(OUT) :: chisq,df,prob,cramrv,ccc
        END SUBROUTINE cntab1
    END INTERFACE
    INTERFACE
        SUBROUTINE cntab2(nn,h,hx,hy,hygx,hxgy,uygx,uxgy,uxy)
        USE nrtype
        INTEGER(I4B), DIMENSION(:,:), INTENT(IN) :: nn
        REAL(SP), INTENT(OUT) :: h,hx,hy,hygx,hxgy,uygx,uxgy,uxy
        END SUBROUTINE cntab2
    END INTERFACE
    INTERFACE
        FUNCTION convlv(data,respns,isign)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: data
        REAL(SP), DIMENSION(:), INTENT(IN) :: respns
        INTEGER(I4B), INTENT(IN) :: isign
        REAL(SP), DIMENSION(size(data)) :: convlv
        END FUNCTION convlv
    END INTERFACE
    INTERFACE
        FUNCTION correl(data1,data2)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
        REAL(SP), DIMENSION(size(data1)) :: correl
        END FUNCTION correl
    END INTERFACE
    INTERFACE
        SUBROUTINE cosft1(y)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
```

```
        END SUBROUTINE cosft1
    END INTERFACE
    INTERFACE
        SUBROUTINE cosft2(y,isign)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE cosft2
    END INTERFACE
    INTERFACE
        SUBROUTINE covsrt(covar,maska)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: covar
        LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: maska
        END SUBROUTINE covsrt
    END INTERFACE
    INTERFACE
        SUBROUTINE cyclic(a,b,c,alpha,beta,r,x)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN):: a,b,c,r
        REAL(SP), INTENT(IN) :: alpha,beta
        REAL(SP), DIMENSION(:), INTENT(OUT):: x
        END SUBROUTINE cyclic
    END INTERFACE
    INTERFACE
        SUBROUTINE daub4(a,isign)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: a
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE daub4
    END INTERFACE
    INTERFACE dawson
        FUNCTION dawson_s(x)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        REAL(SP) :: dawson_s
        END FUNCTION dawson_s
!BL
        FUNCTION dawson_v(x)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x
        REAL(SP), DIMENSION(size(x)) :: dawson_v
        END FUNCTION dawson_v
    END INTERFACE
    INTERFACE
        FUNCTION dbrent(ax,bx,cx,func,dbrent_dfunc,tol,xmin)
```

```
            USE nrtype
            REAL(SP), INTENT(IN) :: ax,bx,cx,tol
            REAL(SP), INTENT(OUT) :: xmin
            REAL(SP) :: dbrent
            INTERFACE
                FUNCTION func(x)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: func
                END FUNCTION func
!BL
                FUNCTION dbrent_dfunc(x)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: dbrent_dfunc
                END FUNCTION dbrent_dfunc
            END INTERFACE
            END FUNCTION dbrent
        END INTERFACE
        INTERFACE
            SUBROUTINE ddpoly(c,x,pd)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP), DIMENSION(:), INTENT(IN) :: c
            REAL(SP), DIMENSION(:), INTENT(OUT) :: pd
            END SUBROUTINE ddpoly
        END INTERFACE
        INTERFACE
            FUNCTION decchk(string,ch)
            USE nrtype
            CHARACTER(1), DIMENSION(:), INTENT(IN) :: string
            CHARACTER(1), INTENT(OUT) :: ch
            LOGICAL(LGT) :: decchk
            END FUNCTION decchk
        END INTERFACE
        INTERFACE
            SUBROUTINE dfpmin(p,gtol,iter,fret,func,dfunc)
            USE nrtype
            INTEGER(I4B), INTENT(OUT) :: iter
            REAL(SP), INTENT(IN) :: gtol
            REAL(SP), INTENT(OUT) :: fret
            REAL(SP), DIMENSION(:), INTENT(INOUT) :: p
            INTERFACE
                FUNCTION func(p)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: p
```

```
                REAL(SP) :: func
                END FUNCTION func
!BL
                FUNCTION dfunc(p)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: p
                REAL(SP), DIMENSION(size(p)) :: dfunc
                END FUNCTION dfunc
            END INTERFACE
            END SUBROUTINE dfpmin
        END INTERFACE
        INTERFACE
            FUNCTION dfridr(func,x,h,err)
            USE nrtype
            REAL(SP), INTENT(IN) :: x,h
            REAL(SP), INTENT(OUT) :: err
            REAL(SP) :: dfridr
            INTERFACE
                FUNCTION func(x)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: func
                END FUNCTION func
            END INTERFACE
            END FUNCTION dfridr
        END INTERFACE
        INTERFACE
            SUBROUTINE dftcor(w,delta,a,b,endpts,corre,corim,corfac)
            USE nrtype
            REAL(SP), INTENT(IN) :: w,delta,a,b
            REAL(SP), INTENT(OUT) :: corre,corim,corfac
            REAL(SP), DIMENSION(:), INTENT(IN) :: endpts
            END SUBROUTINE dftcor
        END INTERFACE
        INTERFACE
            SUBROUTINE dftint(func,a,b,w,cosint,sinint)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,b,w
            REAL(SP), INTENT(OUT) :: cosint,sinint
            INTERFACE
                FUNCTION func(x)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
                REAL(SP), DIMENSION(size(x)) :: func
                END FUNCTION func
            END INTERFACE
```

```
        END SUBROUTINE dftint
    END INTERFACE
    INTERFACE
        SUBROUTINE difeq(k,k1,k2,jsf,is1,isf,indexv,s,y)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: is1,isf,jsf,k,k1,k2
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: indexv
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: s
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: y
        END SUBROUTINE difeq
    END INTERFACE
    INTERFACE
        FUNCTION eclass(lista,listb,n)
        USE nrtype
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: lista,listb
        INTEGER(I4B), INTENT(IN) :: n
        INTEGER(I4B), DIMENSION(n) :: eclass
        END FUNCTION eclass
    END INTERFACE
    INTERFACE
        FUNCTION eclazz(equiv,n)
        USE nrtype
        INTERFACE
            FUNCTION equiv(i,j)
            USE nrtype
            LOGICAL(LGT) :: equiv
            INTEGER(I4B), INTENT(IN) :: i,j
            END FUNCTION equiv
        END INTERFACE
        INTEGER(I4B), INTENT(IN) :: n
        INTEGER(I4B), DIMENSION(n) :: eclazz
        END FUNCTION eclazz
    END INTERFACE
    INTERFACE
        FUNCTION ei(x)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        REAL(SP) :: ei
        END FUNCTION ei
    END INTERFACE
    INTERFACE
        SUBROUTINE eigsrt(d,v)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: d
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: v
        END SUBROUTINE eigsrt
```

```
        END INTERFACE
        INTERFACE elle
                FUNCTION elle_s(phi,ak)
                USE nrtype
                REAL(SP), INTENT(IN) :: phi,ak
                REAL(SP) :: elle_s
                END FUNCTION elle_s
!BL
                FUNCTION elle_v(phi,ak)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: phi,ak
                REAL(SP), DIMENSION(size(phi)) :: elle_v
                END FUNCTION elle_v
        END INTERFACE
        INTERFACE ellf
                FUNCTION ellf_s(phi,ak)
                USE nrtype
                REAL(SP), INTENT(IN) :: phi,ak
                REAL(SP) :: ellf_s
                END FUNCTION ellf_s
!BL
                FUNCTION ellf_v(phi,ak)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: phi,ak
                REAL(SP), DIMENSION(size(phi)) :: ellf_v
                END FUNCTION ellf_v
        END INTERFACE
        INTERFACE ellpi
                FUNCTION ellpi_s(phi,en,ak)
                USE nrtype
                REAL(SP), INTENT(IN) :: phi,en,ak
                REAL(SP) :: ellpi_s
                END FUNCTION ellpi_s
!BL
                FUNCTION ellpi_v(phi,en,ak)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: phi,en,ak
                REAL(SP), DIMENSION(size(phi)) :: ellpi_v
                END FUNCTION ellpi_v
        END INTERFACE
        INTERFACE
                SUBROUTINE elmhes(a)
                USE nrtype
                REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
                END SUBROUTINE elmhes
        END INTERFACE
```

```
        INTERFACE erf
                FUNCTION erf_s(x)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: erf_s
                END FUNCTION erf_s
!BL
                FUNCTION erf_v(x)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
                REAL(SP), DIMENSION(size(x)) :: erf_v
                END FUNCTION erf_v
        END INTERFACE
        INTERFACE erfc
                FUNCTION erfc_s(x)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: erfc_s
                END FUNCTION erfc_s
!BL
                FUNCTION erfc_v(x)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
                REAL(SP), DIMENSION(size(x)) :: erfc_v
                END FUNCTION erfc_v
        END INTERFACE
        INTERFACE erfcc
                FUNCTION erfcc_s(x)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: erfcc_s
                END FUNCTION erfcc_s
!BL
                FUNCTION erfcc_v(x)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
                REAL(SP), DIMENSION(size(x)) :: erfcc_v
                END FUNCTION erfcc_v
        END INTERFACE
        INTERFACE
                SUBROUTINE eulsum(sum,term,jterm)
                USE nrtype
                REAL(SP), INTENT(INOUT) :: sum
                REAL(SP), INTENT(IN) :: term
                INTEGER(I4B), INTENT(IN) :: jterm
                END SUBROUTINE eulsum
```

```
        END INTERFACE
        INTERFACE
                FUNCTION evlmem(fdt,d,xms)
                USE nrtype
                REAL(SP), INTENT(IN) :: fdt,xms
                REAL(SP), DIMENSION(:), INTENT(IN) :: d
                REAL(SP) :: evlmem
                END FUNCTION evlmem
        END INTERFACE
        INTERFACE expdev
                SUBROUTINE expdev_s(harvest)
                USE nrtype
                REAL(SP), INTENT(OUT) :: harvest
                END SUBROUTINE expdev_s
!BL
                SUBROUTINE expdev_v(harvest)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
                END SUBROUTINE expdev_v
        END INTERFACE
        INTERFACE
                FUNCTION expint(n,x)
                USE nrtype
                INTEGER(I4B), INTENT(IN) :: n
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: expint
                END FUNCTION expint
        END INTERFACE
        INTERFACE factln
                FUNCTION factln_s(n)
                USE nrtype
                INTEGER(I4B), INTENT(IN) :: n
                REAL(SP) :: factln_s
                END FUNCTION factln_s
!BL
                FUNCTION factln_v(n)
                USE nrtype
                INTEGER(I4B), DIMENSION(:), INTENT(IN) :: n
                REAL(SP), DIMENSION(size(n)) :: factln_v
                END FUNCTION factln_v
        END INTERFACE
        INTERFACE factrl
                FUNCTION factrl_s(n)
                USE nrtype
                INTEGER(I4B), INTENT(IN) :: n
                REAL(SP) :: factrl_s
```

```
        END FUNCTION factrl_s
!BL
        FUNCTION factrl_v(n)
        USE nrtype
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: n
        REAL(SP), DIMENSION(size(n)) :: factrl_v
        END FUNCTION factrl_v
    END INTERFACE
    INTERFACE
        SUBROUTINE fasper(x,y,ofac,hifac,px,py,jmax,prob)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
        REAL(SP), INTENT(IN) :: ofac,hifac
        INTEGER(I4B), INTENT(OUT) :: jmax
        REAL(SP), INTENT(OUT) :: prob
        REAL(SP), DIMENSION(:), POINTER :: px,py
        END SUBROUTINE fasper
    END INTERFACE
    INTERFACE
        SUBROUTINE fdjac(x,fvec,df)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: fvec
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: x
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: df
        END SUBROUTINE fdjac
    END INTERFACE
    INTERFACE
        SUBROUTINE fgauss(x,a,y,dyda)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x,a
        REAL(SP), DIMENSION(:), INTENT(OUT) :: y
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: dyda
        END SUBROUTINE fgauss
    END INTERFACE
    INTERFACE
        SUBROUTINE fit(x,y,a,b,siga,sigb,chi2,q,sig)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
        REAL(SP), INTENT(OUT) :: a,b,siga,sigb,chi2,q
        REAL(SP), DIMENSION(:), OPTIONAL, INTENT(IN) :: sig
        END SUBROUTINE fit
    END INTERFACE
    INTERFACE
        SUBROUTINE fitexy(x,y,sigx,sigy,a,b,siga,sigb,chi2,q)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,sigx,sigy
```

```
        REAL(SP), INTENT(OUT) :: a,b,siga,sigb,chi2,q
        END SUBROUTINE fitexy
    END INTERFACE
    INTERFACE
        SUBROUTINE fixrts(d)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: d
        END SUBROUTINE fixrts
    END INTERFACE
    INTERFACE
        FUNCTION fleg(x,n)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP), DIMENSION(n) :: fleg
        END FUNCTION fleg
    END INTERFACE
    INTERFACE
        SUBROUTINE flmoon(n,nph,jd,frac)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: n,nph
        INTEGER(I4B), INTENT(OUT) :: jd
        REAL(SP), INTENT(OUT) :: frac
        END SUBROUTINE flmoon
    END INTERFACE
    INTERFACE four1
        SUBROUTINE four1_sp(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE four1_sp
    END INTERFACE
    INTERFACE
        SUBROUTINE four1_alt(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE four1_alt
    END INTERFACE
    INTERFACE
        SUBROUTINE four1_gather(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE four1_gather
    END INTERFACE
```

```
INTERFACE
        SUBROUTINE four2(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:,:), INTENT(INOUT) :: data
        INTEGER(I4B),INTENT(IN) :: isign
        END SUBROUTINE four2
END INTERFACE
INTERFACE
        SUBROUTINE four2_alt(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:,:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE four2_alt
END INTERFACE
INTERFACE
        SUBROUTINE four3(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:,:,:), INTENT(INOUT) :: data
        INTEGER(I4B),INTENT(IN) :: isign
        END SUBROUTINE four3
END INTERFACE
INTERFACE
        SUBROUTINE four3_alt(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:,:,:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE four3_alt
END INTERFACE
INTERFACE
        SUBROUTINE fourcol(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:,:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE fourcol
END INTERFACE
INTERFACE
        SUBROUTINE fourcol_3d(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:,:,:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE fourcol_3d
END INTERFACE
INTERFACE
        SUBROUTINE fourn_gather(data,nn,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:), INTENT(INOUT) :: data
```

```
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: nn
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE fourn_gather
    END INTERFACE
    INTERFACE fourrow
        SUBROUTINE fourrow_sp(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:,:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE fourrow_sp
    END INTERFACE
    INTERFACE
        SUBROUTINE fourrow_3d(data,isign)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:,:,:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE fourrow_3d
    END INTERFACE
    INTERFACE
        FUNCTION fpoly(x,n)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP), DIMENSION(n) :: fpoly
        END FUNCTION fpoly
    END INTERFACE
    INTERFACE
        SUBROUTINE fred2(a,b,t,f,w,g,ak)
        USE nrtype
        REAL(SP), INTENT(IN) :: a,b
        REAL(SP), DIMENSION(:), INTENT(OUT) :: t,f,w
        INTERFACE
            FUNCTION g(t)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: t
            REAL(SP), DIMENSION(size(t)) :: g
            END FUNCTION g
!BL
            FUNCTION ak(t,s)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: t,s
            REAL(SP), DIMENSION(size(t),size(s)) :: ak
            END FUNCTION ak
        END INTERFACE
        END SUBROUTINE fred2
    END INTERFACE
```

```
        INTERFACE
                FUNCTION fredin(x,a,b,t,f,w,g,ak)
                USE nrtype
                REAL(SP), INTENT(IN) :: a,b
                REAL(SP), DIMENSION(:), INTENT(IN) :: x,t,f,w
                REAL(SP), DIMENSION(size(x)) :: fredin
                INTERFACE
                        FUNCTION g(t)
                        USE nrtype
                        REAL(SP), DIMENSION(:), INTENT(IN) :: t
                        REAL(SP), DIMENSION(size(t)) :: g
                        END FUNCTION g
!BL
                        FUNCTION ak(t,s)
                        USE nrtype
                        REAL(SP), DIMENSION(:), INTENT(IN) :: t,s
                        REAL(SP), DIMENSION(size(t),size(s)) :: ak
                        END FUNCTION ak
                END INTERFACE
                END FUNCTION fredin
        END INTERFACE
        INTERFACE
                SUBROUTINE frenel(x,s,c)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP), INTENT(OUT) :: s,c
                END SUBROUTINE frenel
        END INTERFACE
        INTERFACE
                SUBROUTINE frprmn(p,ftol,iter,fret)
                USE nrtype
                INTEGER(I4B), INTENT(OUT) :: iter
                REAL(SP), INTENT(IN) :: ftol
                REAL(SP), INTENT(OUT) :: fret
                REAL(SP), DIMENSION(:), INTENT(INOUT) :: p
                END SUBROUTINE frprmn
        END INTERFACE
        INTERFACE
                SUBROUTINE ftest(data1,data2,f,prob)
                USE nrtype
                REAL(SP), INTENT(OUT) :: f,prob
                REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
                END SUBROUTINE ftest
        END INTERFACE
        INTERFACE
                FUNCTION gamdev(ia)
```

```
            USE nrtype
            INTEGER(I4B), INTENT(IN) :: ia
            REAL(SP) :: gamdev
            END FUNCTION gamdev
        END INTERFACE
        INTERFACE gammln
            FUNCTION gammln_s(xx)
            USE nrtype
            REAL(SP), INTENT(IN) :: xx
            REAL(SP) :: gammln_s
            END FUNCTION gammln_s
!BL
            FUNCTION gammln_v(xx)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: xx
            REAL(SP), DIMENSION(size(xx)) :: gammln_v
            END FUNCTION gammln_v
        END INTERFACE
        INTERFACE gammp
            FUNCTION gammp_s(a,x)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,x
            REAL(SP) :: gammp_s
            END FUNCTION gammp_s
!BL
            FUNCTION gammp_v(a,x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
            REAL(SP), DIMENSION(size(a)) :: gammp_v
            END FUNCTION gammp_v
        END INTERFACE
        INTERFACE gammq
            FUNCTION gammq_s(a,x)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,x
            REAL(SP) :: gammq_s
            END FUNCTION gammq_s
!BL
            FUNCTION gammq_v(a,x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
            REAL(SP), DIMENSION(size(a)) :: gammq_v
            END FUNCTION gammq_v
        END INTERFACE
        INTERFACE gasdev
            SUBROUTINE gasdev_s(harvest)
```

```
            USE nrtype
            REAL(SP), INTENT(OUT) :: harvest
            END SUBROUTINE gasdev_s
!BL
            SUBROUTINE gasdev_v(harvest)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
            END SUBROUTINE gasdev_v
        END INTERFACE
        INTERFACE
            SUBROUTINE gaucof(a,b,amu0,x,w)
            USE nrtype
            REAL(SP), INTENT(IN) :: amu0
            REAL(SP), DIMENSION(:), INTENT(INOUT) :: a,b
            REAL(SP), DIMENSION(:), INTENT(OUT) :: x,w
            END SUBROUTINE gaucof
        END INTERFACE
        INTERFACE
            SUBROUTINE gauher(x,w)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(OUT) :: x,w
            END SUBROUTINE gauher
        END INTERFACE
        INTERFACE
            SUBROUTINE gaujac(x,w,alf,bet)
            USE nrtype
            REAL(SP), INTENT(IN) :: alf,bet
            REAL(SP), DIMENSION(:), INTENT(OUT) :: x,w
            END SUBROUTINE gaujac
        END INTERFACE
        INTERFACE
            SUBROUTINE gaulag(x,w,alf)
            USE nrtype
            REAL(SP), INTENT(IN) :: alf
            REAL(SP), DIMENSION(:), INTENT(OUT) :: x,w
            END SUBROUTINE gaulag
        END INTERFACE
        INTERFACE
            SUBROUTINE gauleg(x1,x2,x,w)
            USE nrtype
            REAL(SP), INTENT(IN) :: x1,x2
            REAL(SP), DIMENSION(:), INTENT(OUT) :: x,w
            END SUBROUTINE gauleg
        END INTERFACE
        INTERFACE
            SUBROUTINE gaussj(a,b)
```

```
            USE nrtype
            REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a,b
            END SUBROUTINE gaussj
        END INTERFACE
        INTERFACE gcf
            FUNCTION gcf_s(a,x,gln)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,x
            REAL(SP), OPTIONAL, INTENT(OUT) :: gln
            REAL(SP) :: gcf_s
            END FUNCTION gcf_s
!BL
            FUNCTION gcf_v(a,x,gln)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
            REAL(SP), DIMENSION(:), OPTIONAL, INTENT(OUT) :: gln
            REAL(SP), DIMENSION(size(a)) :: gcf_v
            END FUNCTION gcf_v
        END INTERFACE
        INTERFACE
            FUNCTION golden(ax,bx,cx,func,tol,xmin)
            USE nrtype
            REAL(SP), INTENT(IN) :: ax,bx,cx,tol
            REAL(SP), INTENT(OUT) :: xmin
            REAL(SP) :: golden
            INTERFACE
                FUNCTION func(x)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: func
                END FUNCTION func
            END INTERFACE
            END FUNCTION golden
        END INTERFACE
        INTERFACE gser
            FUNCTION gser_s(a,x,gln)
            USE nrtype
            REAL(SP), INTENT(IN) :: a,x
            REAL(SP), OPTIONAL, INTENT(OUT) :: gln
            REAL(SP) :: gser_s
            END FUNCTION gser_s
!BL
            FUNCTION gser_v(a,x,gln)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: a,x
            REAL(SP), DIMENSION(:), OPTIONAL, INTENT(OUT) :: gln
```

```
        REAL(SP), DIMENSION(size(a)) :: gser_v
        END FUNCTION gser_v
    END INTERFACE
    INTERFACE
        SUBROUTINE hqr(a,wr,wi)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(OUT) :: wr,wi
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
        END SUBROUTINE hqr
    END INTERFACE
    INTERFACE
        SUBROUTINE hunt(xx,x,jlo)
        USE nrtype
        INTEGER(I4B), INTENT(INOUT) :: jlo
        REAL(SP), INTENT(IN) :: x
        REAL(SP), DIMENSION(:), INTENT(IN) :: xx
        END SUBROUTINE hunt
    END INTERFACE
    INTERFACE
        SUBROUTINE hypdrv(s,ry,rdyds)
        USE nrtype
        REAL(SP), INTENT(IN) :: s
        REAL(SP), DIMENSION(:), INTENT(IN) :: ry
        REAL(SP), DIMENSION(:), INTENT(OUT) :: rdyds
        END SUBROUTINE hypdrv
    END INTERFACE
    INTERFACE
        FUNCTION hypgeo(a,b,c,z)
        USE nrtype
        COMPLEX(SPC), INTENT(IN) :: a,b,c,z
        COMPLEX(SPC) :: hypgeo
        END FUNCTION hypgeo
    END INTERFACE
    INTERFACE
        SUBROUTINE hypser(a,b,c,z,series,deriv)
        USE nrtype
        COMPLEX(SPC), INTENT(IN) :: a,b,c,z
        COMPLEX(SPC), INTENT(OUT) :: series,deriv
        END SUBROUTINE hypser
    END INTERFACE
    INTERFACE
        FUNCTION icrc(crc,buf,jinit,jrev)
        USE nrtype
        CHARACTER(1), DIMENSION(:), INTENT(IN) :: buf
        INTEGER(I2B), INTENT(IN) :: crc,jinit
        INTEGER(I4B), INTENT(IN) :: jrev
```

```
        INTEGER(I2B) :: icrc
        END FUNCTION icrc
    END INTERFACE
    INTERFACE
        FUNCTION igray(n,is)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: n,is
        INTEGER(I4B) :: igray
        END FUNCTION igray
    END INTERFACE
    INTERFACE
        RECURSIVE SUBROUTINE index_bypack(arr,index,partial)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: arr
        INTEGER(I4B), DIMENSION(:), INTENT(INOUT) :: index
        INTEGER, OPTIONAL, INTENT(IN) :: partial
        END SUBROUTINE index_bypack
    END INTERFACE
    INTERFACE indexx
        SUBROUTINE indexx_sp(arr,index)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: arr
        INTEGER(I4B), DIMENSION(:), INTENT(OUT) :: index
        END SUBROUTINE indexx_sp
        SUBROUTINE indexx_i4b(iarr,index)
        USE nrtype
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: iarr
        INTEGER(I4B), DIMENSION(:), INTENT(OUT) :: index
        END SUBROUTINE indexx_i4b
    END INTERFACE
    INTERFACE
        FUNCTION interp(uc)
        USE nrtype
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: uc
        REAL(DP), DIMENSION(2*size(uc,1)-1,2*size(uc,1)-1) :: interp
        END FUNCTION interp
    END INTERFACE
    INTERFACE
        FUNCTION rank(indx)
        USE nrtype
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: indx
        INTEGER(I4B), DIMENSION(size(indx)) :: rank
        END FUNCTION rank
    END INTERFACE
    INTERFACE
        FUNCTION irbit1(iseed)
```

```
        USE nrtype
        INTEGER(I4B), INTENT(INOUT) :: iseed
        INTEGER(I4B) :: irbit1
        END FUNCTION irbit1
    END INTERFACE
    INTERFACE
        FUNCTION irbit2(iseed)
        USE nrtype
        INTEGER(I4B), INTENT(INOUT) :: iseed
        INTEGER(I4B) :: irbit2
        END FUNCTION irbit2
    END INTERFACE
    INTERFACE
        SUBROUTINE jacobi(a,d,v,nrot)
        USE nrtype
        INTEGER(I4B), INTENT(OUT) :: nrot
        REAL(SP), DIMENSION(:), INTENT(OUT) :: d
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: v
        END SUBROUTINE jacobi
    END INTERFACE
    INTERFACE
        SUBROUTINE jacobn(x,y,dfdx,dfdy)
        USE nrtype
        REAL(SP), INTENT(IN) :: x
        REAL(SP), DIMENSION(:), INTENT(IN) :: y
        REAL(SP), DIMENSION(:), INTENT(OUT) :: dfdx
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: dfdy
        END SUBROUTINE jacobn
    END INTERFACE
    INTERFACE
        FUNCTION julday(mm,id,iyyy)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: mm,id,iyyy
        INTEGER(I4B) :: julday
        END FUNCTION julday
    END INTERFACE
    INTERFACE
        SUBROUTINE kendl1(data1,data2,tau,z,prob)
        USE nrtype
        REAL(SP), INTENT(OUT) :: tau,z,prob
        REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
        END SUBROUTINE kendl1
    END INTERFACE
    INTERFACE
        SUBROUTINE kendl2(tab,tau,z,prob)
```

```
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: tab
        REAL(SP), INTENT(OUT) :: tau,z,prob
        END SUBROUTINE kendl2
    END INTERFACE
    INTERFACE
        FUNCTION kermom(y,m)
        USE nrtype
        REAL(DP), INTENT(IN) :: y
        INTEGER(I4B), INTENT(IN) :: m
        REAL(DP), DIMENSION(m) :: kermom
        END FUNCTION kermom
    END INTERFACE
    INTERFACE
        SUBROUTINE ks2d1s(x1,y1,quadvl,d1,prob)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x1,y1
        REAL(SP), INTENT(OUT) :: d1,prob
        INTERFACE
            SUBROUTINE quadvl(x,y,fa,fb,fc,fd)
            USE nrtype
            REAL(SP), INTENT(IN) :: x,y
            REAL(SP), INTENT(OUT) :: fa,fb,fc,fd
            END SUBROUTINE quadvl
        END INTERFACE
        END SUBROUTINE ks2d1s
    END INTERFACE
    INTERFACE
        SUBROUTINE ks2d2s(x1,y1,x2,y2,d,prob)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x1,y1,x2,y2
        REAL(SP), INTENT(OUT) :: d,prob
        END SUBROUTINE ks2d2s
    END INTERFACE
    INTERFACE
        SUBROUTINE ksone(data,func,d,prob)
        USE nrtype
        REAL(SP), INTENT(OUT) :: d,prob
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: data
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: func
            END FUNCTION func
        END INTERFACE
```

```
        END SUBROUTINE ksone
    END INTERFACE
    INTERFACE
        SUBROUTINE kstwo(data1,data2,d,prob)
        USE nrtype
        REAL(SP), INTENT(OUT) :: d,prob
        REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
        END SUBROUTINE kstwo
    END INTERFACE
    INTERFACE
        SUBROUTINE laguer(a,x,its)
        USE nrtype
        INTEGER(I4B), INTENT(OUT) :: its
        COMPLEX(SPC), INTENT(INOUT) :: x
        COMPLEX(SPC), DIMENSION(:), INTENT(IN) :: a
        END SUBROUTINE laguer
    END INTERFACE
    INTERFACE
        SUBROUTINE lfit(x,y,sig,a,maska,covar,chisq,funcs)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,sig
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: a
        LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: maska
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: covar
        REAL(SP), INTENT(OUT) :: chisq
        INTERFACE
            SUBROUTINE funcs(x,arr)
            USE nrtype
            REAL(SP),INTENT(IN) :: x
            REAL(SP), DIMENSION(:), INTENT(OUT) :: arr
            END SUBROUTINE funcs
        END INTERFACE
        END SUBROUTINE lfit
    END INTERFACE
    INTERFACE
        SUBROUTINE linbcg(b,x,itol,tol,itmax,iter,err)
        USE nrtype
        REAL(DP), DIMENSION(:), INTENT(IN) :: b
        REAL(DP), DIMENSION(:), INTENT(INOUT) :: x
        INTEGER(I4B), INTENT(IN) :: itol,itmax
        REAL(DP), INTENT(IN) :: tol
        INTEGER(I4B), INTENT(OUT) :: iter
        REAL(DP), INTENT(OUT) :: err
        END SUBROUTINE linbcg
    END INTERFACE
    INTERFACE
```

```
        SUBROUTINE linmin(p,xi,fret)
        USE nrtype
        REAL(SP), INTENT(OUT) :: fret
        REAL(SP), DIMENSION(:), TARGET, INTENT(INOUT) :: p,xi
        END SUBROUTINE linmin
    END INTERFACE
    INTERFACE
        SUBROUTINE lnsrch(xold,fold,g,p,x,f,stpmax,check,func)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: xold,g
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: p
        REAL(SP), INTENT(IN) :: fold,stpmax
        REAL(SP), DIMENSION(:), INTENT(OUT) :: x
        REAL(SP), INTENT(OUT) :: f
        LOGICAL(LGT), INTENT(OUT) :: check
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP) :: func
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            END FUNCTION func
        END INTERFACE
        END SUBROUTINE lnsrch
    END INTERFACE
    INTERFACE
        FUNCTION locate(xx,x)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: xx
        REAL(SP), INTENT(IN) :: x
        INTEGER(I4B) :: locate
        END FUNCTION locate
    END INTERFACE
    INTERFACE
        FUNCTION lop(u)
        USE nrtype
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: u
        REAL(DP), DIMENSION(size(u,1),size(u,1)) :: lop
        END FUNCTION lop
    END INTERFACE
    INTERFACE
        SUBROUTINE lubksb(a,indx,b)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: a
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: indx
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: b
        END SUBROUTINE lubksb
```

```
END INTERFACE
INTERFACE
	SUBROUTINE ludcmp(a,indx,d)
	USE nrtype
	REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
	INTEGER(I4B), DIMENSION(:), INTENT(OUT) :: indx
	REAL(SP), INTENT(OUT) :: d
	END SUBROUTINE ludcmp
END INTERFACE
INTERFACE
	SUBROUTINE machar(ibeta,it,irnd,ngrd,machep,negep,iexp,minexp,&
		maxexp,eps,epsneg,xmin,xmax)
	USE nrtype
	INTEGER(I4B), INTENT(OUT) :: ibeta,iexp,irnd,it,machep,maxexp,&
		minexp,negep,ngrd
	REAL(SP), INTENT(OUT) :: eps,epsneg,xmax,xmin
	END SUBROUTINE machar
END INTERFACE
INTERFACE
	SUBROUTINE medfit(x,y,a,b,abdev)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
	REAL(SP), INTENT(OUT) :: a,b,abdev
	END SUBROUTINE medfit
END INTERFACE
INTERFACE
	SUBROUTINE memcof(data,xms,d)
	USE nrtype
	REAL(SP), INTENT(OUT) :: xms
	REAL(SP), DIMENSION(:), INTENT(IN) :: data
	REAL(SP), DIMENSION(:), INTENT(OUT) :: d
	END SUBROUTINE memcof
END INTERFACE
INTERFACE
	SUBROUTINE mgfas(u,maxcyc)
	USE nrtype
	REAL(DP), DIMENSION(:,:), INTENT(INOUT) :: u
	INTEGER(I4B), INTENT(IN) :: maxcyc
	END SUBROUTINE mgfas
END INTERFACE
INTERFACE
	SUBROUTINE mglin(u,ncycle)
	USE nrtype
	REAL(DP), DIMENSION(:,:), INTENT(INOUT) :: u
	INTEGER(I4B), INTENT(IN) :: ncycle
	END SUBROUTINE mglin
```

```
END INTERFACE
INTERFACE
	SUBROUTINE midexp(funk,aa,bb,s,n)
	USE nrtype
	REAL(SP), INTENT(IN) :: aa,bb
	REAL(SP), INTENT(INOUT) :: s
	INTEGER(I4B), INTENT(IN) :: n
	INTERFACE
		FUNCTION funk(x)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		REAL(SP), DIMENSION(size(x)) :: funk
		END FUNCTION funk
	END INTERFACE
	END SUBROUTINE midexp
END INTERFACE
INTERFACE
	SUBROUTINE midinf(funk,aa,bb,s,n)
	USE nrtype
	REAL(SP), INTENT(IN) :: aa,bb
	REAL(SP), INTENT(INOUT) :: s
	INTEGER(I4B), INTENT(IN) :: n
	INTERFACE
		FUNCTION funk(x)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		REAL(SP), DIMENSION(size(x)) :: funk
		END FUNCTION funk
	END INTERFACE
	END SUBROUTINE midinf
END INTERFACE
INTERFACE
	SUBROUTINE midpnt(func,a,b,s,n)
	USE nrtype
	REAL(SP), INTENT(IN) :: a,b
	REAL(SP), INTENT(INOUT) :: s
	INTEGER(I4B), INTENT(IN) :: n
	INTERFACE
		FUNCTION func(x)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		REAL(SP), DIMENSION(size(x)) :: func
		END FUNCTION func
	END INTERFACE
	END SUBROUTINE midpnt
END INTERFACE
```

```
INTERFACE
	SUBROUTINE midsql(funk,aa,bb,s,n)
	USE nrtype
	REAL(SP), INTENT(IN) :: aa,bb
	REAL(SP), INTENT(INOUT) :: s
	INTEGER(I4B), INTENT(IN) :: n
	INTERFACE
		FUNCTION funk(x)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		REAL(SP), DIMENSION(size(x)) :: funk
		END FUNCTION funk
	END INTERFACE
	END SUBROUTINE midsql
END INTERFACE
INTERFACE
	SUBROUTINE midsqu(funk,aa,bb,s,n)
	USE nrtype
	REAL(SP), INTENT(IN) :: aa,bb
	REAL(SP), INTENT(INOUT) :: s
	INTEGER(I4B), INTENT(IN) :: n
	INTERFACE
		FUNCTION funk(x)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		REAL(SP), DIMENSION(size(x)) :: funk
		END FUNCTION funk
	END INTERFACE
	END SUBROUTINE midsqu
END INTERFACE
INTERFACE
	RECURSIVE SUBROUTINE miser(func,regn,ndim,npts,dith,ave,var)
	USE nrtype
	INTERFACE
		FUNCTION func(x)
		USE nrtype
		REAL(SP) :: func
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		END FUNCTION func
	END INTERFACE
	REAL(SP), DIMENSION(:), INTENT(IN) :: regn
	INTEGER(I4B), INTENT(IN) :: ndim,npts
	REAL(SP), INTENT(IN) :: dith
	REAL(SP), INTENT(OUT) :: ave,var
	END SUBROUTINE miser
END INTERFACE
```

```
INTERFACE
	SUBROUTINE mmid(y,dydx,xs,htot,nstep,yout,derivs)
	USE nrtype
	INTEGER(I4B), INTENT(IN) :: nstep
	REAL(SP), INTENT(IN) :: xs,htot
	REAL(SP), DIMENSION(:), INTENT(IN) :: y,dydx
	REAL(SP), DIMENSION(:), INTENT(OUT) :: yout
	INTERFACE
		SUBROUTINE derivs(x,y,dydx)
		USE nrtype
		REAL(SP), INTENT(IN) :: x
		REAL(SP), DIMENSION(:), INTENT(IN) :: y
		REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
		END SUBROUTINE derivs
	END INTERFACE
	END SUBROUTINE mmid
END INTERFACE
INTERFACE
	SUBROUTINE mnbrak(ax,bx,cx,fa,fb,fc,func)
	USE nrtype
	REAL(SP), INTENT(INOUT) :: ax,bx
	REAL(SP), INTENT(OUT) :: cx,fa,fb,fc
	INTERFACE
		FUNCTION func(x)
		USE nrtype
		REAL(SP), INTENT(IN) :: x
		REAL(SP) :: func
		END FUNCTION func
	END INTERFACE
	END SUBROUTINE mnbrak
END INTERFACE
INTERFACE
	SUBROUTINE mnewt(ntrial,x,tolx,tolf,usrfun)
	USE nrtype
	INTEGER(I4B), INTENT(IN) :: ntrial
	REAL(SP), INTENT(IN) :: tolx,tolf
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: x
	INTERFACE
		SUBROUTINE usrfun(x,fvec,fjac)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		REAL(SP), DIMENSION(:), INTENT(OUT) :: fvec
		REAL(SP), DIMENSION(:,:), INTENT(OUT) :: fjac
		END SUBROUTINE usrfun
	END INTERFACE
	END SUBROUTINE mnewt
```

```
END INTERFACE
INTERFACE
	SUBROUTINE moment(data,ave,adev,sdev,var,skew,curt)
	USE nrtype
	REAL(SP), INTENT(OUT) :: ave,adev,sdev,var,skew,curt
	REAL(SP), DIMENSION(:), INTENT(IN) :: data
	END SUBROUTINE moment
END INTERFACE
INTERFACE
	SUBROUTINE mp2dfr(a,s,n,m)
	USE nrtype
	INTEGER(I4B), INTENT(IN) :: n
	INTEGER(I4B), INTENT(OUT) :: m
	CHARACTER(1), DIMENSION(:), INTENT(INOUT) :: a
	CHARACTER(1), DIMENSION(:), INTENT(OUT) :: s
	END SUBROUTINE mp2dfr
END INTERFACE
INTERFACE
	SUBROUTINE mpdiv(q,r,u,v,n,m)
	USE nrtype
	CHARACTER(1), DIMENSION(:), INTENT(OUT) :: q,r
	CHARACTER(1), DIMENSION(:), INTENT(IN) :: u,v
	INTEGER(I4B), INTENT(IN) :: n,m
	END SUBROUTINE mpdiv
END INTERFACE
INTERFACE
	SUBROUTINE mpinv(u,v,n,m)
	USE nrtype
	CHARACTER(1), DIMENSION(:), INTENT(OUT) :: u
	CHARACTER(1), DIMENSION(:), INTENT(IN) :: v
	INTEGER(I4B), INTENT(IN) :: n,m
	END SUBROUTINE mpinv
END INTERFACE
INTERFACE
	SUBROUTINE mpmul(w,u,v,n,m)
	USE nrtype
	CHARACTER(1), DIMENSION(:), INTENT(IN) :: u,v
	CHARACTER(1), DIMENSION(:), INTENT(OUT) :: w
	INTEGER(I4B), INTENT(IN) :: n,m
	END SUBROUTINE mpmul
END INTERFACE
INTERFACE
	SUBROUTINE mppi(n)
	USE nrtype
	INTEGER(I4B), INTENT(IN) :: n
	END SUBROUTINE mppi
```

```
END INTERFACE
INTERFACE
	SUBROUTINE mprove(a,alud,indx,b,x)
	USE nrtype
	REAL(SP), DIMENSION(:,:), INTENT(IN) :: a,alud
	INTEGER(I4B), DIMENSION(:), INTENT(IN) :: indx
	REAL(SP), DIMENSION(:), INTENT(IN) :: b
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: x
	END SUBROUTINE mprove
END INTERFACE
INTERFACE
	SUBROUTINE mpsqrt(w,u,v,n,m)
	USE nrtype
	CHARACTER(1), DIMENSION(:), INTENT(OUT) :: w,u
	CHARACTER(1), DIMENSION(:), INTENT(IN) :: v
	INTEGER(I4B), INTENT(IN) :: n,m
	END SUBROUTINE mpsqrt
END INTERFACE
INTERFACE
	SUBROUTINE mrqcof(x,y,sig,a,maska,alpha,beta,chisq,funcs)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,a,sig
	REAL(SP), DIMENSION(:), INTENT(OUT) :: beta
	REAL(SP), DIMENSION(:,:), INTENT(OUT) :: alpha
	REAL(SP), INTENT(OUT) :: chisq
	LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: maska
	INTERFACE
		SUBROUTINE funcs(x,a,yfit,dyda)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x,a
		REAL(SP), DIMENSION(:), INTENT(OUT) :: yfit
		REAL(SP), DIMENSION(:,:), INTENT(OUT) :: dyda
		END SUBROUTINE funcs
	END INTERFACE
	END SUBROUTINE mrqcof
END INTERFACE
INTERFACE
	SUBROUTINE mrqmin(x,y,sig,a,maska,covar,alpha,chisq,funcs,alamda)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,sig
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: a
	REAL(SP), DIMENSION(:,:), INTENT(OUT) :: covar,alpha
	REAL(SP), INTENT(OUT) :: chisq
	REAL(SP), INTENT(INOUT) :: alamda
	LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: maska
	INTERFACE
```

```
                SUBROUTINE funcs(x,a,yfit,dyda)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x,a
                REAL(SP), DIMENSION(:), INTENT(OUT) :: yfit
                REAL(SP), DIMENSION(:,:), INTENT(OUT) :: dyda
                END SUBROUTINE funcs
            END INTERFACE
            END SUBROUTINE mrqmin
        END INTERFACE
        INTERFACE
            SUBROUTINE newt(x,check)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(INOUT) :: x
            LOGICAL(LGT), INTENT(OUT) :: check
            END SUBROUTINE newt
        END INTERFACE
        INTERFACE
            SUBROUTINE odeint(ystart,x1,x2,eps,h1,hmin,derivs,rkqs)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(INOUT) :: ystart
            REAL(SP), INTENT(IN) :: x1,x2,eps,h1,hmin
            INTERFACE
                SUBROUTINE derivs(x,y,dydx)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP), DIMENSION(:), INTENT(IN) :: y
                REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
                END SUBROUTINE derivs
!BL
                SUBROUTINE rkqs(y,dydx,x,htry,eps,yscal,hdid,hnext,derivs)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
                REAL(SP), DIMENSION(:), INTENT(IN) :: dydx,yscal
                REAL(SP), INTENT(INOUT) :: x
                REAL(SP), INTENT(IN) :: htry,eps
                REAL(SP), INTENT(OUT) :: hdid,hnext
                    INTERFACE
                    SUBROUTINE derivs(x,y,dydx)
                        USE nrtype
                        REAL(SP), INTENT(IN) :: x
                        REAL(SP), DIMENSION(:), INTENT(IN) :: y
                        REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
                        END SUBROUTINE derivs
                    END INTERFACE
                END SUBROUTINE rkqs
            END INTERFACE
```

```
        END SUBROUTINE odeint
    END INTERFACE
    INTERFACE
        SUBROUTINE orthog(anu,alpha,beta,a,b)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: anu,alpha,beta
        REAL(SP), DIMENSION(:), INTENT(OUT) :: a,b
        END SUBROUTINE orthog
    END INTERFACE
    INTERFACE
        SUBROUTINE pade(cof,resid)
        USE nrtype
        REAL(DP), DIMENSION(:), INTENT(INOUT) :: cof
        REAL(SP), INTENT(OUT) :: resid
        END SUBROUTINE pade
    END INTERFACE
    INTERFACE
        FUNCTION pccheb(d)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: d
        REAL(SP), DIMENSION(size(d)) :: pccheb
        END FUNCTION pccheb
    END INTERFACE
    INTERFACE
        SUBROUTINE pcshft(a,b,d)
        USE nrtype
        REAL(SP), INTENT(IN) :: a,b
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: d
        END SUBROUTINE pcshft
    END INTERFACE
    INTERFACE
        SUBROUTINE pearsn(x,y,r,prob,z)
        USE nrtype
        REAL(SP), INTENT(OUT) :: r,prob,z
        REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
        END SUBROUTINE pearsn
    END INTERFACE
    INTERFACE
        SUBROUTINE period(x,y,ofac,hifac,px,py,jmax,prob)
        USE nrtype
        INTEGER(I4B), INTENT(OUT) :: jmax
        REAL(SP), INTENT(IN) :: ofac,hifac
        REAL(SP), INTENT(OUT) :: prob
        REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
        REAL(SP), DIMENSION(:), POINTER :: px,py
        END SUBROUTINE period
```

```
        END INTERFACE
        INTERFACE plgndr
                FUNCTION plgndr_s(l,m,x)
                USE nrtype
                INTEGER(I4B), INTENT(IN) :: l,m
                REAL(SP), INTENT(IN) :: x
                REAL(SP) :: plgndr_s
                END FUNCTION plgndr_s
!BL
                FUNCTION plgndr_v(l,m,x)
                USE nrtype
                INTEGER(I4B), INTENT(IN) :: l,m
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
                REAL(SP), DIMENSION(size(x)) :: plgndr_v
                END FUNCTION plgndr_v
        END INTERFACE
        INTERFACE
                FUNCTION poidev(xm)
                USE nrtype
                REAL(SP), INTENT(IN) :: xm
                REAL(SP) :: poidev
                END FUNCTION poidev
        END INTERFACE
        INTERFACE
                FUNCTION polcoe(x,y)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
                REAL(SP), DIMENSION(size(x)) :: polcoe
                END FUNCTION polcoe
        END INTERFACE
        INTERFACE
                FUNCTION polcof(xa,ya)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: xa,ya
                REAL(SP), DIMENSION(size(xa)) :: polcof
                END FUNCTION polcof
        END INTERFACE
        INTERFACE
                SUBROUTINE poldiv(u,v,q,r)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: u,v
                REAL(SP), DIMENSION(:), INTENT(OUT) :: q,r
                END SUBROUTINE poldiv
        END INTERFACE
        INTERFACE
                SUBROUTINE polin2(x1a,x2a,ya,x1,x2,y,dy)
```

```
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x1a,x2a
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: ya
        REAL(SP), INTENT(IN) :: x1,x2
        REAL(SP), INTENT(OUT) :: y,dy
        END SUBROUTINE polin2
    END INTERFACE
    INTERFACE
        SUBROUTINE polint(xa,ya,x,y,dy)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: xa,ya
        REAL(SP), INTENT(IN) :: x
        REAL(SP), INTENT(OUT) :: y,dy
        END SUBROUTINE polint
    END INTERFACE
    INTERFACE
        SUBROUTINE powell(p,xi,ftol,iter,fret)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: p
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: xi
        INTEGER(I4B), INTENT(OUT) :: iter
        REAL(SP), INTENT(IN) :: ftol
        REAL(SP), INTENT(OUT) :: fret
        END SUBROUTINE powell
    END INTERFACE
    INTERFACE
        FUNCTION predic(data,d,nfut)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: data,d
        INTEGER(I4B), INTENT(IN) :: nfut
        REAL(SP), DIMENSION(nfut) :: predic
        END FUNCTION predic
    END INTERFACE
    INTERFACE
        FUNCTION probks(alam)
        USE nrtype
        REAL(SP), INTENT(IN) :: alam
        REAL(SP) :: probks
        END FUNCTION probks
    END INTERFACE
    INTERFACE psdes
        SUBROUTINE psdes_s(lword,rword)
        USE nrtype
        INTEGER(I4B), INTENT(INOUT) :: lword,rword
        END SUBROUTINE psdes_s
!BL
```

```
        SUBROUTINE psdes_v(lword,rword)
        USE nrtype
        INTEGER(I4B), DIMENSION(:), INTENT(INOUT) :: lword,rword
        END SUBROUTINE psdes_v
    END INTERFACE
    INTERFACE
        SUBROUTINE pwt(a,isign)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: a
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE pwt
    END INTERFACE
    INTERFACE
        SUBROUTINE pwtset(n)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: n
        END SUBROUTINE pwtset
    END INTERFACE
    INTERFACE pythag
        FUNCTION pythag_sp(a,b)
        USE nrtype
        REAL(SP), INTENT(IN) :: a,b
        REAL(SP) :: pythag_sp
        END FUNCTION pythag_sp
    END INTERFACE
    INTERFACE
        SUBROUTINE pzextr(iest,xest,yest,yz,dy)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: iest
        REAL(SP), INTENT(IN) :: xest
        REAL(SP), DIMENSION(:), INTENT(IN) :: yest
        REAL(SP), DIMENSION(:), INTENT(OUT) :: yz,dy
        END SUBROUTINE pzextr
    END INTERFACE
    INTERFACE
        SUBROUTINE qrdcmp(a,c,d,sing)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
        REAL(SP), DIMENSION(:), INTENT(OUT) :: c,d
        LOGICAL(LGT), INTENT(OUT) :: sing
        END SUBROUTINE qrdcmp
    END INTERFACE
    INTERFACE
        FUNCTION qromb(func,a,b)
        USE nrtype
        REAL(SP), INTENT(IN) :: a,b
```

```
        REAL(SP) :: qromb
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: func
            END FUNCTION func
        END INTERFACE
        END FUNCTION qromb
    END INTERFACE
    INTERFACE
        FUNCTION qromo(func,a,b,choose)
        USE nrtype
        REAL(SP), INTENT(IN) :: a,b
        REAL(SP) :: qromo
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x
            REAL(SP), DIMENSION(size(x)) :: func
            END FUNCTION func
        END INTERFACE
        INTERFACE
            SUBROUTINE choose(funk,aa,bb,s,n)
            USE nrtype
            REAL(SP), INTENT(IN) :: aa,bb
            REAL(SP), INTENT(INOUT) :: s
            INTEGER(I4B), INTENT(IN) :: n
            INTERFACE
                FUNCTION funk(x)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
                REAL(SP), DIMENSION(size(x)) :: funk
                END FUNCTION funk
            END INTERFACE
            END SUBROUTINE choose
        END INTERFACE
        END FUNCTION qromo
    END INTERFACE
    INTERFACE
        SUBROUTINE qroot(p,b,c,eps)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: p
        REAL(SP), INTENT(INOUT) :: b,c
        REAL(SP), INTENT(IN) :: eps
        END SUBROUTINE qroot
```

```
END INTERFACE
INTERFACE
	SUBROUTINE qrsolv(a,c,d,b)
	USE nrtype
	REAL(SP), DIMENSION(:,:), INTENT(IN) :: a
	REAL(SP), DIMENSION(:), INTENT(IN) :: c,d
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: b
	END SUBROUTINE qrsolv
END INTERFACE
INTERFACE
	SUBROUTINE qrupdt(r,qt,u,v)
	USE nrtype
	REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: r,qt
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: u
	REAL(SP), DIMENSION(:), INTENT(IN) :: v
	END SUBROUTINE qrupdt
END INTERFACE
INTERFACE
	FUNCTION qsimp(func,a,b)
	USE nrtype
	REAL(SP), INTENT(IN) :: a,b
	REAL(SP) :: qsimp
	INTERFACE
		FUNCTION func(x)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		REAL(SP), DIMENSION(size(x)) :: func
		END FUNCTION func
	END INTERFACE
	END FUNCTION qsimp
END INTERFACE
INTERFACE
	FUNCTION qtrap(func,a,b)
	USE nrtype
	REAL(SP), INTENT(IN) :: a,b
	REAL(SP) :: qtrap
	INTERFACE
		FUNCTION func(x)
		USE nrtype
		REAL(SP), DIMENSION(:), INTENT(IN) :: x
		REAL(SP), DIMENSION(size(x)) :: func
		END FUNCTION func
	END INTERFACE
	END FUNCTION qtrap
END INTERFACE
INTERFACE
```

```
        SUBROUTINE quadct(x,y,xx,yy,fa,fb,fc,fd)
        USE nrtype
        REAL(SP), INTENT(IN) :: x,y
        REAL(SP), DIMENSION(:), INTENT(IN) :: xx,yy
        REAL(SP), INTENT(OUT) :: fa,fb,fc,fd
        END SUBROUTINE quadct
    END INTERFACE
    INTERFACE
        SUBROUTINE quadmx(a)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: a
        END SUBROUTINE quadmx
    END INTERFACE
    INTERFACE
        SUBROUTINE quadvl(x,y,fa,fb,fc,fd)
        USE nrtype
        REAL(SP), INTENT(IN) :: x,y
        REAL(SP), INTENT(OUT) :: fa,fb,fc,fd
        END SUBROUTINE quadvl
    END INTERFACE
    INTERFACE
        FUNCTION ran(idum)
        INTEGER(selected_int_kind(9)), INTENT(INOUT) :: idum
        REAL :: ran
        END FUNCTION ran
    END INTERFACE
    INTERFACE ran0
        SUBROUTINE ran0_s(harvest)
        USE nrtype
        REAL(SP), INTENT(OUT) :: harvest
        END SUBROUTINE ran0_s
!BL
        SUBROUTINE ran0_v(harvest)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
        END SUBROUTINE ran0_v
    END INTERFACE
    INTERFACE ran1
        SUBROUTINE ran1_s(harvest)
        USE nrtype
        REAL(SP), INTENT(OUT) :: harvest
        END SUBROUTINE ran1_s
!BL
        SUBROUTINE ran1_v(harvest)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
```

```
        END SUBROUTINE ran1_v
    END INTERFACE
    INTERFACE ran2
        SUBROUTINE ran2_s(harvest)
        USE nrtype
        REAL(SP), INTENT(OUT) :: harvest
        END SUBROUTINE ran2_s
!BL
        SUBROUTINE ran2_v(harvest)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
        END SUBROUTINE ran2_v
    END INTERFACE
    INTERFACE ran3
        SUBROUTINE ran3_s(harvest)
        USE nrtype
        REAL(SP), INTENT(OUT) :: harvest
        END SUBROUTINE ran3_s
!BL
        SUBROUTINE ran3_v(harvest)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
        END SUBROUTINE ran3_v
    END INTERFACE
    INTERFACE
        SUBROUTINE ratint(xa,ya,x,y,dy)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: xa,ya
        REAL(SP), INTENT(IN) :: x
        REAL(SP), INTENT(OUT) :: y,dy
        END SUBROUTINE ratint
    END INTERFACE
    INTERFACE
        SUBROUTINE ratlsq(func,a,b,mm,kk,cof,dev)
        USE nrtype
        REAL(DP), INTENT(IN) :: a,b
        INTEGER(I4B), INTENT(IN) :: mm,kk
        REAL(DP), DIMENSION(:), INTENT(OUT) :: cof
        REAL(DP), INTENT(OUT) :: dev
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(DP), DIMENSION(:), INTENT(IN) :: x
            REAL(DP), DIMENSION(size(x)) :: func
            END FUNCTION func
        END INTERFACE
```

```
            END SUBROUTINE ratlsq
        END INTERFACE
        INTERFACE ratval
            FUNCTION ratval_s(x,cof,mm,kk)
            USE nrtype
            REAL(DP), INTENT(IN) :: x
            INTEGER(I4B), INTENT(IN) :: mm,kk
            REAL(DP), DIMENSION(mm+kk+1), INTENT(IN) :: cof
            REAL(DP) :: ratval_s
            END FUNCTION ratval_s
!BL
            FUNCTION ratval_v(x,cof,mm,kk)
            USE nrtype
            REAL(DP), DIMENSION(:), INTENT(IN) :: x
            INTEGER(I4B), INTENT(IN) :: mm,kk
            REAL(DP), DIMENSION(mm+kk+1), INTENT(IN) :: cof
            REAL(DP), DIMENSION(size(x)) :: ratval_v
            END FUNCTION ratval_v
        END INTERFACE
        INTERFACE rc
            FUNCTION rc_s(x,y)
            USE nrtype
            REAL(SP), INTENT(IN) :: x,y
            REAL(SP) :: rc_s
            END FUNCTION rc_s
!BL
            FUNCTION rc_v(x,y)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
            REAL(SP), DIMENSION(size(x)) :: rc_v
            END FUNCTION rc_v
        END INTERFACE
        INTERFACE rd
            FUNCTION rd_s(x,y,z)
            USE nrtype
            REAL(SP), INTENT(IN) :: x,y,z
            REAL(SP) :: rd_s
            END FUNCTION rd_s
!BL
            FUNCTION rd_v(x,y,z)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,z
            REAL(SP), DIMENSION(size(x)) :: rd_v
            END FUNCTION rd_v
        END INTERFACE
        INTERFACE realft
```

```
        SUBROUTINE realft_sp(data,isign,zdata)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: data
        INTEGER(I4B), INTENT(IN) :: isign
        COMPLEX(SPC), DIMENSION(:), OPTIONAL, TARGET :: zdata
        END SUBROUTINE realft_sp
    END INTERFACE
    INTERFACE
        RECURSIVE FUNCTION recur1(a,b) RESULT(u)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: a,b
        REAL(SP), DIMENSION(size(a)) :: u
        END FUNCTION recur1
    END INTERFACE
    INTERFACE
        FUNCTION recur2(a,b,c)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: a,b,c
        REAL(SP), DIMENSION(size(a)) :: recur2
        END FUNCTION recur2
    END INTERFACE
    INTERFACE
        SUBROUTINE relax(u,rhs)
        USE nrtype
        REAL(DP), DIMENSION(:,:), INTENT(INOUT) :: u
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: rhs
        END SUBROUTINE relax
    END INTERFACE
    INTERFACE
        SUBROUTINE relax2(u,rhs)
        USE nrtype
        REAL(DP), DIMENSION(:,:), INTENT(INOUT) :: u
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: rhs
        END SUBROUTINE relax2
    END INTERFACE
    INTERFACE
    FUNCTION resid(u,rhs)
        USE nrtype
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: u,rhs
        REAL(DP), DIMENSION(size(u,1),size(u,1)) :: resid
        END FUNCTION resid
    END INTERFACE
    INTERFACE rf
        FUNCTION rf_s(x,y,z)
        USE nrtype
        REAL(SP), INTENT(IN) :: x,y,z
```

```
            REAL(SP) :: rf_s
            END FUNCTION rf_s
!BL
            FUNCTION rf_v(x,y,z)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,z
            REAL(SP), DIMENSION(size(x)) :: rf_v
            END FUNCTION rf_v
        END INTERFACE
        INTERFACE rj
            FUNCTION rj_s(x,y,z,p)
            USE nrtype
            REAL(SP), INTENT(IN) :: x,y,z,p
            REAL(SP) :: rj_s
            END FUNCTION rj_s
!BL
            FUNCTION rj_v(x,y,z,p)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,z,p
            REAL(SP), DIMENSION(size(x)) :: rj_v
            END FUNCTION rj_v
        END INTERFACE
        INTERFACE
            SUBROUTINE rk4(y,dydx,x,h,yout,derivs)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: y,dydx
            REAL(SP), INTENT(IN) :: x,h
            REAL(SP), DIMENSION(:), INTENT(OUT) :: yout
            INTERFACE
                SUBROUTINE derivs(x,y,dydx)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP), DIMENSION(:), INTENT(IN) :: y
                REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
                END SUBROUTINE derivs
            END INTERFACE
            END SUBROUTINE rk4
        END INTERFACE
        INTERFACE
            SUBROUTINE rkck(y,dydx,x,h,yout,yerr,derivs)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: y,dydx
            REAL(SP), INTENT(IN) :: x,h
            REAL(SP), DIMENSION(:), INTENT(OUT) :: yout,yerr
            INTERFACE
                SUBROUTINE derivs(x,y,dydx)
```

```
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP), DIMENSION(:), INTENT(IN) :: y
            REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
            END SUBROUTINE derivs
        END INTERFACE
        END SUBROUTINE rkck
    END INTERFACE
    INTERFACE
        SUBROUTINE rkdumb(vstart,x1,x2,nstep,derivs)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: vstart
        REAL(SP), INTENT(IN) :: x1,x2
        INTEGER(I4B), INTENT(IN) :: nstep
        INTERFACE
            SUBROUTINE derivs(x,y,dydx)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP), DIMENSION(:), INTENT(IN) :: y
            REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
            END SUBROUTINE derivs
        END INTERFACE
        END SUBROUTINE rkdumb
    END INTERFACE
    INTERFACE
        SUBROUTINE rkqs(y,dydx,x,htry,eps,yscal,hdid,hnext,derivs)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
        REAL(SP), DIMENSION(:), INTENT(IN) :: dydx,yscal
        REAL(SP), INTENT(INOUT) :: x
        REAL(SP), INTENT(IN) :: htry,eps
        REAL(SP), INTENT(OUT) :: hdid,hnext
        INTERFACE
            SUBROUTINE derivs(x,y,dydx)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP), DIMENSION(:), INTENT(IN) :: y
            REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
            END SUBROUTINE derivs
        END INTERFACE
        END SUBROUTINE rkqs
    END INTERFACE
    INTERFACE
        SUBROUTINE rlft2(data,spec,speq,isign)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: data
```

```
        COMPLEX(SPC), DIMENSION(:,:), INTENT(OUT) :: spec
        COMPLEX(SPC), DIMENSION(:), INTENT(OUT) :: speq
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE rlft2
    END INTERFACE
    INTERFACE
        SUBROUTINE rlft3(data,spec,speq,isign)
        USE nrtype
        REAL(SP), DIMENSION(:,:,:), INTENT(INOUT) :: data
        COMPLEX(SPC), DIMENSION(:,:,:), INTENT(OUT) :: spec
        COMPLEX(SPC), DIMENSION(:,:), INTENT(OUT) :: speq
        INTEGER(I4B), INTENT(IN) :: isign
        END SUBROUTINE rlft3
    END INTERFACE
    INTERFACE
        SUBROUTINE rotate(r,qt,i,a,b)
        USE nrtype
        REAL(SP), DIMENSION(:,:), TARGET, INTENT(INOUT) :: r,qt
        INTEGER(I4B), INTENT(IN) :: i
        REAL(SP), INTENT(IN) :: a,b
        END SUBROUTINE rotate
    END INTERFACE
    INTERFACE
        SUBROUTINE rsolv(a,d,b)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: a
        REAL(SP), DIMENSION(:), INTENT(IN) :: d
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: b
        END SUBROUTINE rsolv
    END INTERFACE
    INTERFACE
        FUNCTION rstrct(uf)
        USE nrtype
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: uf
        REAL(DP), DIMENSION((size(uf,1)+1)/2,(size(uf,1)+1)/2) :: rstrct
        END FUNCTION rstrct
    END INTERFACE
    INTERFACE
        FUNCTION rtbis(func,x1,x2,xacc)
        USE nrtype
        REAL(SP), INTENT(IN) :: x1,x2,xacc
        REAL(SP) :: rtbis
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
```

```
            REAL(SP) :: func
            END FUNCTION func
        END INTERFACE
        END FUNCTION rtbis
    END INTERFACE
    INTERFACE
        FUNCTION rtflsp(func,x1,x2,xacc)
        USE nrtype
        REAL(SP), INTENT(IN) :: x1,x2,xacc
        REAL(SP) :: rtflsp
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: func
            END FUNCTION func
        END INTERFACE
        END FUNCTION rtflsp
    END INTERFACE
    INTERFACE
        FUNCTION rtnewt(funcd,x1,x2,xacc)
        USE nrtype
        REAL(SP), INTENT(IN) :: x1,x2,xacc
        REAL(SP) :: rtnewt
        INTERFACE
            SUBROUTINE funcd(x,fval,fderiv)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP), INTENT(OUT) :: fval,fderiv
            END SUBROUTINE funcd
        END INTERFACE
        END FUNCTION rtnewt
    END INTERFACE
    INTERFACE
        FUNCTION rtsafe(funcd,x1,x2,xacc)
        USE nrtype
        REAL(SP), INTENT(IN) :: x1,x2,xacc
        REAL(SP) :: rtsafe
        INTERFACE
            SUBROUTINE funcd(x,fval,fderiv)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP), INTENT(OUT) :: fval,fderiv
            END SUBROUTINE funcd
        END INTERFACE
        END FUNCTION rtsafe
```

```
END INTERFACE
INTERFACE
	FUNCTION rtsec(func,x1,x2,xacc)
	USE nrtype
	REAL(SP), INTENT(IN) :: x1,x2,xacc
	REAL(SP) :: rtsec
	INTERFACE
		FUNCTION func(x)
		USE nrtype
		REAL(SP), INTENT(IN) :: x
		REAL(SP) :: func
		END FUNCTION func
	END INTERFACE
	END FUNCTION rtsec
END INTERFACE
INTERFACE
	SUBROUTINE rzextr(iest,xest,yest,yz,dy)
	USE nrtype
	INTEGER(I4B), INTENT(IN) :: iest
	REAL(SP), INTENT(IN) :: xest
	REAL(SP), DIMENSION(:), INTENT(IN) :: yest
	REAL(SP), DIMENSION(:), INTENT(OUT) :: yz,dy
	END SUBROUTINE rzextr
END INTERFACE
INTERFACE
	FUNCTION savgol(nl,nrr,ld,m)
	USE nrtype
	INTEGER(I4B), INTENT(IN) :: nl,nrr,ld,m
	REAL(SP), DIMENSION(nl+nrr+1) :: savgol
	END FUNCTION savgol
END INTERFACE
INTERFACE
	SUBROUTINE scrsho(func)
	USE nrtype
	INTERFACE
		FUNCTION func(x)
		USE nrtype
		REAL(SP), INTENT(IN) :: x
		REAL(SP) :: func
		END FUNCTION func
	END INTERFACE
	END SUBROUTINE scrsho
END INTERFACE
INTERFACE
	FUNCTION select(k,arr)
	USE nrtype
```

```
        INTEGER(I4B), INTENT(IN) :: k
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
        REAL(SP) :: select
        END FUNCTION select
    END INTERFACE
    INTERFACE
        FUNCTION select_bypack(k,arr)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: k
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
        REAL(SP) :: select_bypack
        END FUNCTION select_bypack
    END INTERFACE
    INTERFACE
        SUBROUTINE select_heap(arr,heap)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: arr
        REAL(SP), DIMENSION(:), INTENT(OUT) :: heap
        END SUBROUTINE select_heap
    END INTERFACE
    INTERFACE
        FUNCTION select_inplace(k,arr)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: k
        REAL(SP), DIMENSION(:), INTENT(IN) :: arr
        REAL(SP) :: select_inplace
        END FUNCTION select_inplace
    END INTERFACE
    INTERFACE
        SUBROUTINE simplx(a,m1,m2,m3,icase,izrov,iposv)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
        INTEGER(I4B), INTENT(IN) :: m1,m2,m3
        INTEGER(I4B), INTENT(OUT) :: icase
        INTEGER(I4B), DIMENSION(:), INTENT(OUT) :: izrov,iposv
        END SUBROUTINE simplx
    END INTERFACE
    INTERFACE
        SUBROUTINE simpr(y,dydx,dfdx,dfdy,xs,htot,nstep,yout,derivs)
        USE nrtype
        REAL(SP), INTENT(IN) :: xs,htot
        REAL(SP), DIMENSION(:), INTENT(IN) :: y,dydx,dfdx
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: dfdy
        INTEGER(I4B), INTENT(IN) :: nstep
        REAL(SP), DIMENSION(:), INTENT(OUT) :: yout
        INTERFACE
```

```
            SUBROUTINE derivs(x,y,dydx)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP), DIMENSION(:), INTENT(IN) :: y
            REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
            END SUBROUTINE derivs
        END INTERFACE
        END SUBROUTINE simpr
    END INTERFACE
    INTERFACE
        SUBROUTINE sinft(y)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
        END SUBROUTINE sinft
    END INTERFACE
    INTERFACE
        SUBROUTINE slvsm2(u,rhs)
        USE nrtype
        REAL(DP), DIMENSION(3,3), INTENT(OUT) :: u
        REAL(DP), DIMENSION(3,3), INTENT(IN) :: rhs
        END SUBROUTINE slvsm2
    END INTERFACE
    INTERFACE
        SUBROUTINE slvsml(u,rhs)
        USE nrtype
        REAL(DP), DIMENSION(3,3), INTENT(OUT) :: u
        REAL(DP), DIMENSION(3,3), INTENT(IN) :: rhs
        END SUBROUTINE slvsml
    END INTERFACE
    INTERFACE
        SUBROUTINE sncndn(uu,emmc,sn,cn,dn)
        USE nrtype
        REAL(SP), INTENT(IN) :: uu,emmc
        REAL(SP), INTENT(OUT) :: sn,cn,dn
        END SUBROUTINE sncndn
    END INTERFACE
    INTERFACE
        FUNCTION snrm(sx,itol)
        USE nrtype
        REAL(DP), DIMENSION(:), INTENT(IN) :: sx
        INTEGER(I4B), INTENT(IN) :: itol
        REAL(DP) :: snrm
        END FUNCTION snrm
    END INTERFACE
    INTERFACE
        SUBROUTINE sobseq(x,init)
```

```
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(OUT) :: x
        INTEGER(I4B), OPTIONAL, INTENT(IN) :: init
        END SUBROUTINE sobseq
    END INTERFACE
    INTERFACE
        SUBROUTINE solvde(itmax,conv,slowc,scalv,indexv,nb,y)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: itmax,nb
        REAL(SP), INTENT(IN) :: conv,slowc
        REAL(SP), DIMENSION(:), INTENT(IN) :: scalv
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: indexv
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: y
        END SUBROUTINE solvde
    END INTERFACE
    INTERFACE
        SUBROUTINE sor(a,b,c,d,e,f,u,rjac)
        USE nrtype
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: a,b,c,d,e,f
        REAL(DP), DIMENSION(:,:), INTENT(INOUT) :: u
        REAL(DP), INTENT(IN) :: rjac
        END SUBROUTINE sor
    END INTERFACE
    INTERFACE
        SUBROUTINE sort(arr)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
        END SUBROUTINE sort
    END INTERFACE
    INTERFACE
        SUBROUTINE sort2(arr,slave)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr,slave
        END SUBROUTINE sort2
    END INTERFACE
    INTERFACE
        SUBROUTINE sort3(arr,slave1,slave2)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr,slave1,slave2
        END SUBROUTINE sort3
    END INTERFACE
    INTERFACE
        SUBROUTINE sort_bypack(arr)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
        END SUBROUTINE sort_bypack
```

```
END INTERFACE
INTERFACE
	SUBROUTINE sort_byreshape(arr)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
	END SUBROUTINE sort_byreshape
END INTERFACE
INTERFACE
	SUBROUTINE sort_heap(arr)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
	END SUBROUTINE sort_heap
END INTERFACE
INTERFACE
	SUBROUTINE sort_pick(arr)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
	END SUBROUTINE sort_pick
END INTERFACE
INTERFACE
	SUBROUTINE sort_radix(arr)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
	END SUBROUTINE sort_radix
END INTERFACE
INTERFACE
	SUBROUTINE sort_shell(arr)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
	END SUBROUTINE sort_shell
END INTERFACE
INTERFACE
	SUBROUTINE spctrm(p,k,ovrlap,unit,n_window)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(OUT) :: p
	INTEGER(I4B), INTENT(IN) :: k
	LOGICAL(LGT), INTENT(IN) :: ovrlap
	INTEGER(I4B), OPTIONAL, INTENT(IN) :: n_window,unit
	END SUBROUTINE spctrm
END INTERFACE
INTERFACE
	SUBROUTINE spear(data1,data2,d,zd,probd,rs,probrs)
	USE nrtype
	REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
	REAL(SP), INTENT(OUT) :: d,zd,probd,rs,probrs
	END SUBROUTINE spear
```

```
        END INTERFACE
        INTERFACE sphbes
                SUBROUTINE sphbes_s(n,x,sj,sy,sjp,syp)
                USE nrtype
                INTEGER(I4B), INTENT(IN) :: n
                REAL(SP), INTENT(IN) :: x
                REAL(SP), INTENT(OUT) :: sj,sy,sjp,syp
                END SUBROUTINE sphbes_s
!BL
                SUBROUTINE sphbes_v(n,x,sj,sy,sjp,syp)
                USE nrtype
                INTEGER(I4B), INTENT(IN) :: n
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
                REAL(SP), DIMENSION(:), INTENT(OUT) :: sj,sy,sjp,syp
                END SUBROUTINE sphbes_v
        END INTERFACE
        INTERFACE
                SUBROUTINE splie2(x1a,x2a,ya,y2a)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x1a,x2a
                REAL(SP), DIMENSION(:,:), INTENT(IN) :: ya
                REAL(SP), DIMENSION(:,:), INTENT(OUT) :: y2a
                END SUBROUTINE splie2
        END INTERFACE
        INTERFACE
                FUNCTION splin2(x1a,x2a,ya,y2a,x1,x2)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x1a,x2a
                REAL(SP), DIMENSION(:,:), INTENT(IN) :: ya,y2a
                REAL(SP), INTENT(IN) :: x1,x2
                REAL(SP) :: splin2
                END FUNCTION splin2
        END INTERFACE
        INTERFACE
                SUBROUTINE spline(x,y,yp1,ypn,y2)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x,y
                REAL(SP), INTENT(IN) :: yp1,ypn
                REAL(SP), DIMENSION(:), INTENT(OUT) :: y2
                END SUBROUTINE spline
        END INTERFACE
        INTERFACE
                FUNCTION splint(xa,ya,y2a,x)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: xa,ya,y2a
                REAL(SP), INTENT(IN) :: x
```

```
        REAL(SP) :: splint
        END FUNCTION splint
    END INTERFACE
    INTERFACE sprsax
        SUBROUTINE sprsax_dp(sa,x,b)
        USE nrtype
        TYPE(sprs2_dp), INTENT(IN) :: sa
        REAL(DP), DIMENSION (:), INTENT(IN) :: x
        REAL(DP), DIMENSION (:), INTENT(OUT) :: b
        END SUBROUTINE sprsax_dp
!BL
        SUBROUTINE sprsax_sp(sa,x,b)
        USE nrtype
        TYPE(sprs2_sp), INTENT(IN) :: sa
        REAL(SP), DIMENSION (:), INTENT(IN) :: x
        REAL(SP), DIMENSION (:), INTENT(OUT) :: b
        END SUBROUTINE sprsax_sp
    END INTERFACE
    INTERFACE sprsdiag
        SUBROUTINE sprsdiag_dp(sa,b)
        USE nrtype
        TYPE(sprs2_dp), INTENT(IN) :: sa
        REAL(DP), DIMENSION(:), INTENT(OUT) :: b
        END SUBROUTINE sprsdiag_dp
!BL
        SUBROUTINE sprsdiag_sp(sa,b)
        USE nrtype
        TYPE(sprs2_sp), INTENT(IN) :: sa
        REAL(SP), DIMENSION(:), INTENT(OUT) :: b
        END SUBROUTINE sprsdiag_sp
    END INTERFACE
    INTERFACE sprsin
        SUBROUTINE sprsin_sp(a,thresh,sa)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: a
        REAL(SP), INTENT(IN) :: thresh
        TYPE(sprs2_sp), INTENT(OUT) :: sa
        END SUBROUTINE sprsin_sp
!BL
        SUBROUTINE sprsin_dp(a,thresh,sa)
        USE nrtype
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: a
        REAL(DP), INTENT(IN) :: thresh
        TYPE(sprs2_dp), INTENT(OUT) :: sa
        END SUBROUTINE sprsin_dp
    END INTERFACE
```

```
        INTERFACE
                SUBROUTINE sprstp(sa)
                USE nrtype
                TYPE(sprs2_sp), INTENT(INOUT) :: sa
                END SUBROUTINE sprstp
        END INTERFACE
        INTERFACE sprstx
                SUBROUTINE sprstx_dp(sa,x,b)
                USE nrtype
                TYPE(sprs2_dp), INTENT(IN) :: sa
                REAL(DP), DIMENSION (:), INTENT(IN) :: x
                REAL(DP), DIMENSION (:), INTENT(OUT) :: b
                END SUBROUTINE sprstx_dp
!BL
                SUBROUTINE sprstx_sp(sa,x,b)
                USE nrtype
                TYPE(sprs2_sp), INTENT(IN) :: sa
                REAL(SP), DIMENSION (:), INTENT(IN) :: x
                REAL(SP), DIMENSION (:), INTENT(OUT) :: b
                END SUBROUTINE sprstx_sp
        END INTERFACE
        INTERFACE
                SUBROUTINE stifbs(y,dydx,x,htry,eps,yscal,hdid,hnext,derivs)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
                REAL(SP), DIMENSION(:), INTENT(IN) :: dydx,yscal
                REAL(SP), INTENT(IN) :: htry,eps
                REAL(SP), INTENT(INOUT) :: x
                REAL(SP), INTENT(OUT) :: hdid,hnext
                INTERFACE
                        SUBROUTINE derivs(x,y,dydx)
                        USE nrtype
                        REAL(SP), INTENT(IN) :: x
                        REAL(SP), DIMENSION(:), INTENT(IN) :: y
                        REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
                        END SUBROUTINE derivs
                END INTERFACE
                END SUBROUTINE stifbs
        END INTERFACE
        INTERFACE
                SUBROUTINE stiff(y,dydx,x,htry,eps,yscal,hdid,hnext,derivs)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(INOUT) :: y
                REAL(SP), DIMENSION(:), INTENT(IN) :: dydx,yscal
                REAL(SP), INTENT(INOUT) :: x
                REAL(SP), INTENT(IN) :: htry,eps
```

```
        REAL(SP), INTENT(OUT) :: hdid,hnext
        INTERFACE
                SUBROUTINE derivs(x,y,dydx)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP), DIMENSION(:), INTENT(IN) :: y
                REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
                END SUBROUTINE derivs
        END INTERFACE
        END SUBROUTINE stiff
END INTERFACE
INTERFACE
        SUBROUTINE stoerm(y,d2y,xs,htot,nstep,yout,derivs)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: y,d2y
        REAL(SP), INTENT(IN) :: xs,htot
        INTEGER(I4B), INTENT(IN) :: nstep
        REAL(SP), DIMENSION(:), INTENT(OUT) :: yout
        INTERFACE
                SUBROUTINE derivs(x,y,dydx)
                USE nrtype
                REAL(SP), INTENT(IN) :: x
                REAL(SP), DIMENSION(:), INTENT(IN) :: y
                REAL(SP), DIMENSION(:), INTENT(OUT) :: dydx
                END SUBROUTINE derivs
        END INTERFACE
        END SUBROUTINE stoerm
END INTERFACE
INTERFACE svbksb
        SUBROUTINE svbksb_sp(u,w,v,b,x)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: u,v
        REAL(SP), DIMENSION(:), INTENT(IN) :: w,b
        REAL(SP), DIMENSION(:), INTENT(OUT) :: x
        END SUBROUTINE svbksb_sp
END INTERFACE
INTERFACE svdcmp
        SUBROUTINE svdcmp_sp(a,w,v)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
        REAL(SP), DIMENSION(:), INTENT(OUT) :: w
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: v
        END SUBROUTINE svdcmp_sp
END INTERFACE
INTERFACE
        SUBROUTINE svdfit(x,y,sig,a,v,w,chisq,funcs)
```

```
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: x,y,sig
        REAL(SP), DIMENSION(:), INTENT(OUT) :: a,w
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: v
        REAL(SP), INTENT(OUT) :: chisq
        INTERFACE
            FUNCTION funcs(x,n)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            INTEGER(I4B), INTENT(IN) :: n
            REAL(SP), DIMENSION(n) :: funcs
            END FUNCTION funcs
        END INTERFACE
        END SUBROUTINE svdfit
    END INTERFACE
    INTERFACE
        SUBROUTINE svdvar(v,w,cvm)
        USE nrtype
        REAL(SP), DIMENSION(:,:), INTENT(IN) :: v
        REAL(SP), DIMENSION(:), INTENT(IN) :: w
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: cvm
        END SUBROUTINE svdvar
    END INTERFACE
    INTERFACE
        FUNCTION toeplz(r,y)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: r,y
        REAL(SP), DIMENSION(size(y)) :: toeplz
        END FUNCTION toeplz
    END INTERFACE
    INTERFACE
        SUBROUTINE tptest(data1,data2,t,prob)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
        REAL(SP), INTENT(OUT) :: t,prob
        END SUBROUTINE tptest
    END INTERFACE
    INTERFACE
        SUBROUTINE tqli(d,e,z)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: d,e
        REAL(SP), DIMENSION(:,:), OPTIONAL, INTENT(INOUT) :: z
        END SUBROUTINE tqli
    END INTERFACE
    INTERFACE
        SUBROUTINE trapzd(func,a,b,s,n)
```

```
            USE nrtype
            REAL(SP), INTENT(IN) :: a,b
            REAL(SP), INTENT(INOUT) :: s
            INTEGER(I4B), INTENT(IN) :: n
            INTERFACE
                FUNCTION func(x)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(IN) :: x
                REAL(SP), DIMENSION(size(x)) :: func
                END FUNCTION func
            END INTERFACE
            END SUBROUTINE trapzd
        END INTERFACE
        INTERFACE
            SUBROUTINE tred2(a,d,e,novectors)
            USE nrtype
            REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
            REAL(SP), DIMENSION(:), INTENT(OUT) :: d,e
            LOGICAL(LGT), OPTIONAL, INTENT(IN) :: novectors
            END SUBROUTINE tred2
        END INTERFACE
!       On a purely serial machine, for greater efficiency, remove
!       the generic name tridag from the following interface,
!       and put it on the next one after that.
        INTERFACE tridag
            RECURSIVE SUBROUTINE tridag_par(a,b,c,r,u)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: a,b,c,r
            REAL(SP), DIMENSION(:), INTENT(OUT) :: u
            END SUBROUTINE tridag_par
        END INTERFACE
        INTERFACE
            SUBROUTINE tridag_ser(a,b,c,r,u)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: a,b,c,r
            REAL(SP), DIMENSION(:), INTENT(OUT) :: u
            END SUBROUTINE tridag_ser
        END INTERFACE
        INTERFACE
            SUBROUTINE ttest(data1,data2,t,prob)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
            REAL(SP), INTENT(OUT) :: t,prob
            END SUBROUTINE ttest
        END INTERFACE
        INTERFACE
```

```
        SUBROUTINE tutest(data1,data2,t,prob)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
        REAL(SP), INTENT(OUT) :: t,prob
        END SUBROUTINE tutest
    END INTERFACE
    INTERFACE
        SUBROUTINE twofft(data1,data2,fft1,fft2)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: data1,data2
        COMPLEX(SPC), DIMENSION(:), INTENT(OUT) :: fft1,fft2
        END SUBROUTINE twofft
    END INTERFACE
    INTERFACE
        FUNCTION vander(x,q)
        USE nrtype
        REAL(DP), DIMENSION(:), INTENT(IN) :: x,q
        REAL(DP), DIMENSION(size(x)) :: vander
        END FUNCTION vander
    END INTERFACE
    INTERFACE
        SUBROUTINE vegas(region,func,init,ncall,itmx,nprn,tgral,sd,chi2a)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: region
        INTEGER(I4B), INTENT(IN) :: init,ncall,itmx,nprn
        REAL(SP), INTENT(OUT) :: tgral,sd,chi2a
        INTERFACE
            FUNCTION func(pt,wgt)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(IN) :: pt
            REAL(SP), INTENT(IN) :: wgt
            REAL(SP) :: func
            END FUNCTION func
        END INTERFACE
        END SUBROUTINE vegas
    END INTERFACE
    INTERFACE
        SUBROUTINE voltra(t0,h,t,f,g,ak)
        USE nrtype
        REAL(SP), INTENT(IN) :: t0,h
        REAL(SP), DIMENSION(:), INTENT(OUT) :: t
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: f
        INTERFACE
            FUNCTION g(t)
            USE nrtype
            REAL(SP), INTENT(IN) :: t
```

```
                REAL(SP), DIMENSION(:), POINTER :: g
                END FUNCTION g
!BL
                FUNCTION ak(t,s)
                USE nrtype
                REAL(SP), INTENT(IN) :: t,s
                REAL(SP), DIMENSION(:,:), POINTER :: ak
                END FUNCTION ak
            END INTERFACE
            END SUBROUTINE voltra
        END INTERFACE
        INTERFACE
            SUBROUTINE wt1(a,isign,wtstep)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(INOUT) :: a
            INTEGER(I4B), INTENT(IN) :: isign
            INTERFACE
                SUBROUTINE wtstep(a,isign)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(INOUT) :: a
                INTEGER(I4B), INTENT(IN) :: isign
                END SUBROUTINE wtstep
            END INTERFACE
            END SUBROUTINE wt1
        END INTERFACE
        INTERFACE
            SUBROUTINE wtn(a,nn,isign,wtstep)
            USE nrtype
            REAL(SP), DIMENSION(:), INTENT(INOUT) :: a
            INTEGER(I4B), DIMENSION(:), INTENT(IN) :: nn
            INTEGER(I4B), INTENT(IN) :: isign
            INTERFACE
                SUBROUTINE wtstep(a,isign)
                USE nrtype
                REAL(SP), DIMENSION(:), INTENT(INOUT) :: a
                INTEGER(I4B), INTENT(IN) :: isign
                END SUBROUTINE wtstep
            END INTERFACE
            END SUBROUTINE wtn
        END INTERFACE
        INTERFACE
            FUNCTION wwghts(n,h,kermom)
            USE nrtype
            INTEGER(I4B), INTENT(IN) :: n
            REAL(SP), INTENT(IN) :: h
            REAL(SP), DIMENSION(n) :: wwghts
```

```
        INTERFACE
            FUNCTION kermom(y,m)
            USE nrtype
            REAL(DP), INTENT(IN) :: y
            INTEGER(I4B), INTENT(IN) :: m
            REAL(DP), DIMENSION(m) :: kermom
            END FUNCTION kermom
        END INTERFACE
        END FUNCTION wwghts
    END INTERFACE
    INTERFACE
        SUBROUTINE zbrac(func,x1,x2,succes)
        USE nrtype
        REAL(SP), INTENT(INOUT) :: x1,x2
        LOGICAL(LGT), INTENT(OUT) :: succes
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: func
            END FUNCTION func
        END INTERFACE
        END SUBROUTINE zbrac
    END INTERFACE
    INTERFACE
        SUBROUTINE zbrak(func,x1,x2,n,xb1,xb2,nb)
        USE nrtype
        INTEGER(I4B), INTENT(IN) :: n
        INTEGER(I4B), INTENT(OUT) :: nb
        REAL(SP), INTENT(IN) :: x1,x2
        REAL(SP), DIMENSION(:), POINTER :: xb1,xb2
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: func
            END FUNCTION func
        END INTERFACE
        END SUBROUTINE zbrak
    END INTERFACE
    INTERFACE
        FUNCTION zbrent(func,x1,x2,tol)
        USE nrtype
        REAL(SP), INTENT(IN) :: x1,x2,tol
        REAL(SP) :: zbrent
        INTERFACE
```

```
            FUNCTION func(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: func
            END FUNCTION func
        END INTERFACE
        END FUNCTION zbrent
    END INTERFACE
    INTERFACE
        SUBROUTINE zrhqr(a,rtr,rti)
        USE nrtype
        REAL(SP), DIMENSION(:), INTENT(IN) :: a
        REAL(SP), DIMENSION(:), INTENT(OUT) :: rtr,rti
        END SUBROUTINE zrhqr
    END INTERFACE
    INTERFACE
        FUNCTION zriddr(func,x1,x2,xacc)
        USE nrtype
        REAL(SP), INTENT(IN) :: x1,x2,xacc
        REAL(SP) :: zriddr
        INTERFACE
            FUNCTION func(x)
            USE nrtype
            REAL(SP), INTENT(IN) :: x
            REAL(SP) :: func
            END FUNCTION func
        END INTERFACE
        END FUNCTION zriddr
    END INTERFACE
    INTERFACE
        SUBROUTINE zroots(a,roots,polish)
        USE nrtype
        COMPLEX(SPC), DIMENSION(:), INTENT(IN) :: a
        COMPLEX(SPC), DIMENSION(:), INTENT(OUT) :: roots
        LOGICAL(LGT), INTENT(IN) :: polish
        END SUBROUTINE zroots
    END INTERFACE
END MODULE nr
```

```
MODULE nrtype
	INTEGER, PARAMETER :: I4B = SELECTED_INT_KIND(9)
	INTEGER, PARAMETER :: I2B = SELECTED_INT_KIND(4)
	INTEGER, PARAMETER :: I1B = SELECTED_INT_KIND(2)
	INTEGER, PARAMETER :: SP = KIND(1.0D0)
	INTEGER, PARAMETER :: DP = KIND(1.0D0)
	INTEGER, PARAMETER :: SPC = KIND((1.0D0,1.0D0))
	INTEGER, PARAMETER :: DPC = KIND((1.0D0,1.0D0))
	INTEGER, PARAMETER :: LGT = KIND(.true.)
	REAL(SP), PARAMETER :: PI=3.141592653589793238462643383279502884197_sp
	REAL(SP), PARAMETER :: PIO2=1.570796326794896619231321691639751442098 58_sp
	REAL(SP), PARAMETER :: TWOPI=6.283185307179586476925286766559005768394_sp
	REAL(SP), PARAMETER :: SQRT2=1.414213562373095048801688724209698078569 67_sp
	REAL(SP), PARAMETER :: EULER=0.577215664901532860606512090082402431042 2_sp
	REAL(DP), PARAMETER :: PI_D=3.141592653589793238462643383279502884197_dp
	REAL(DP), PARAMETER :: PIO2_D=1.570796326794896619231321691639751442098 58_dp
	REAL(DP), PARAMETER :: TWOPI_D=6.283185307179586476925286766559005768394_dp
	TYPE sprs2_sp
		INTEGER(I4B) :: n,len
		REAL(SP), DIMENSION(:), POINTER :: val
		INTEGER(I4B), DIMENSION(:), POINTER :: irow
		INTEGER(I4B), DIMENSION(:), POINTER :: jcol
	END TYPE sprs2_sp
	TYPE sprs2_dp
		INTEGER(I4B) :: n,len
		REAL(DP), DIMENSION(:), POINTER :: val
		INTEGER(I4B), DIMENSION(:), POINTER :: irow
		INTEGER(I4B), DIMENSION(:), POINTER :: jcol
	END TYPE sprs2_dp
END MODULE nrtype
```

```
MODULE nrutil
	USE nrtype
	IMPLICIT NONE
	INTEGER(I4B), PARAMETER :: NPAR_ARTH=16,NPAR2_ARTH=8
	INTEGER(I4B), PARAMETER :: NPAR_GEOP=4,NPAR2_GEOP=2
	INTEGER(I4B), PARAMETER :: NPAR_CUMSUM=16
	INTEGER(I4B), PARAMETER :: NPAR_CUMPROD=8
	INTEGER(I4B), PARAMETER :: NPAR_POLY=8
	INTEGER(I4B), PARAMETER :: NPAR_POLYTERM=8
	INTERFACE array_copy
		MODULE PROCEDURE array_copy_r, array_copy_i
	END INTERFACE
	INTERFACE swap
		MODULE PROCEDURE swap_i,swap_r,swap_rv,swap_c, &
			swap_cv,swap_cm, &
			masked_swap_rs,masked_swap_rv,masked_swap_rm
	END INTERFACE
	INTERFACE reallocate
		MODULE PROCEDURE reallocate_rv,reallocate_rm,&
			reallocate_iv,reallocate_im,reallocate_hv
	END INTERFACE
	INTERFACE imaxloc
		MODULE PROCEDURE imaxloc_r,imaxloc_i
	END INTERFACE
	INTERFACE assert
		MODULE PROCEDURE assert1,assert2,assert3,assert4,assert_v
	END INTERFACE
	INTERFACE assert_eq
		MODULE PROCEDURE assert_eq2,assert_eq3,assert_eq4,assert_eqn
	END INTERFACE
	INTERFACE arth
		MODULE PROCEDURE arth_r, arth_i
	END INTERFACE
	INTERFACE geop
		MODULE PROCEDURE geop_r, geop_i, geop_c, geop_dv
	END INTERFACE
	INTERFACE cumsum
		MODULE PROCEDURE cumsum_r,cumsum_i
	END INTERFACE
	INTERFACE poly
		MODULE PROCEDURE poly_rr,poly_rrv,&
			poly_rc,poly_cc,poly_msk_rrv
	END INTERFACE
	INTERFACE poly_term
		MODULE PROCEDURE poly_term_rr,poly_term_cc
	END INTERFACE
```

```
        INTERFACE outerprod
                MODULE PROCEDURE outerprod_r
        END INTERFACE
        INTERFACE outerdiff
                MODULE PROCEDURE outerdiff_r,outerdiff_i
        END INTERFACE
        INTERFACE scatter_add
                MODULE PROCEDURE scatter_add_r
        END INTERFACE
        INTERFACE scatter_max
                MODULE PROCEDURE scatter_max_r
        END INTERFACE
        INTERFACE diagadd
                MODULE PROCEDURE diagadd_rv,diagadd_r
        END INTERFACE
        INTERFACE diagmult
                MODULE PROCEDURE diagmult_rv,diagmult_r
        END INTERFACE
        INTERFACE get_diag
                MODULE PROCEDURE get_diag_rv
        END INTERFACE
        INTERFACE put_diag
                MODULE PROCEDURE put_diag_rv, put_diag_r
        END INTERFACE
CONTAINS
!BL
        SUBROUTINE array_copy_r(src,dest,n_copied,n_not_copied)
        REAL(SP), DIMENSION(:), INTENT(IN) :: src
        REAL(SP), DIMENSION(:), INTENT(OUT) :: dest
        INTEGER(I4B), INTENT(OUT) :: n_copied, n_not_copied
        n_copied=MIN(SIZE(src),SIZE(dest))
        n_not_copied=SIZE(src)-n_copied
        dest(1:n_copied)=src(1:n_copied)
        END SUBROUTINE array_copy_r
!BL
        SUBROUTINE array_copy_d(src,dest,n_copied,n_not_copied)
        REAL(DP), DIMENSION(:), INTENT(IN) :: src
        REAL(DP), DIMENSION(:), INTENT(OUT) :: dest
        INTEGER(I4B), INTENT(OUT) :: n_copied, n_not_copied
        n_copied=MIN(SIZE(src),SIZE(dest))
        n_not_copied=SIZE(src)-n_copied
        dest(1:n_copied)=src(1:n_copied)
        END SUBROUTINE array_copy_d
!BL
        SUBROUTINE array_copy_i(src,dest,n_copied,n_not_copied)
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: src
```

```
        INTEGER(I4B), DIMENSION(:), INTENT(OUT) :: dest
        INTEGER(I4B), INTENT(OUT) :: n_copied, n_not_copied
        n_copied=MIN(SIZE(src),SIZE(dest))
        n_not_copied=SIZE(src)-n_copied
        dest(1:n_copied)=src(1:n_copied)
        END SUBROUTINE array_copy_i
!BL
!BL
        SUBROUTINE swap_i(a,b)
        INTEGER(I4B), INTENT(INOUT) :: a,b
        INTEGER(I4B) :: dum
        dum=a
        a=b
        b=dum
        END SUBROUTINE swap_i
!BL
        SUBROUTINE swap_r(a,b)
        REAL(SP), INTENT(INOUT) :: a,b
        REAL(SP) :: dum
        dum=a
        a=b
        b=dum
        END SUBROUTINE swap_r
!BL
        SUBROUTINE swap_rv(a,b)
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: a,b
        REAL(SP), DIMENSION(SIZE(a)) :: dum
        dum=a
        a=b
        b=dum
        END SUBROUTINE swap_rv
!BL
        SUBROUTINE swap_c(a,b)
        COMPLEX(SPC), INTENT(INOUT) :: a,b
        COMPLEX(SPC) :: dum
        dum=a
        a=b
        b=dum
        END SUBROUTINE swap_c
!BL
        SUBROUTINE swap_cv(a,b)
        COMPLEX(SPC), DIMENSION(:), INTENT(INOUT) :: a,b
        COMPLEX(SPC), DIMENSION(SIZE(a)) :: dum
        dum=a
        a=b
        b=dum
```

```
        END SUBROUTINE swap_cv
!BL
        SUBROUTINE swap_cm(a,b)
        COMPLEX(SPC), DIMENSION(:,:), INTENT(INOUT) :: a,b
        COMPLEX(SPC), DIMENSION(SIZE(a,1),SIZE(a,2)) :: dum
        dum=a
        a=b
        b=dum
        END SUBROUTINE swap_cm
!BL
        SUBROUTINE masked_swap_rs(a,b,mask)
        REAL(SP), INTENT(INOUT) :: a,b
        LOGICAL(LGT), INTENT(IN) :: mask
        REAL(SP) :: swp
        IF (mask) THEN
                swp=a
                a=b
                b=swp
        END IF
        END SUBROUTINE masked_swap_rs
!BL
        SUBROUTINE masked_swap_rv(a,b,mask)
        REAL(SP), DIMENSION(:), INTENT(INOUT) :: a,b
        LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: mask
        REAL(SP), DIMENSION(SIZE(a)) :: swp
        WHERE (mask)
                swp=a
                a=b
                b=swp
        END WHERE
        END SUBROUTINE masked_swap_rv
!BL
        SUBROUTINE masked_swap_rm(a,b,mask)
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a,b
        LOGICAL(LGT), DIMENSION(:,:), INTENT(IN) :: mask
        REAL(SP), DIMENSION(SIZE(a,1),SIZE(a,2)) :: swp
        WHERE (mask)
                swp=a
                a=b
                b=swp
        END WHERE
        END SUBROUTINE masked_swap_rm
!BL
!BL
        FUNCTION reallocate_rv(p,n)
        REAL(SP), DIMENSION(:), POINTER :: p, reallocate_rv
```

```
	INTEGER(I4B), INTENT(IN) :: n
	INTEGER(I4B) :: nold,ierr
	ALLOCATE(reallocate_rv(n),stat=ierr)
	IF (ierr /= 0) CALL &
		nrerror('reallocate_rv: problem in attempt to allocate memory')
	IF (.NOT. ASSOCIATED(p)) RETURN
	nold=SIZE(p)
	reallocate_rv(1:MIN(nold,n))=p(1:MIN(nold,n))
	DEALLOCATE(p)
	END FUNCTION reallocate_rv
!BL
	FUNCTION reallocate_iv(p,n)
	INTEGER(I4B), DIMENSION(:), POINTER :: p, reallocate_iv
	INTEGER(I4B), INTENT(IN) :: n
	INTEGER(I4B) :: nold,ierr
	ALLOCATE(reallocate_iv(n),stat=ierr)
	IF (ierr /= 0) CALL &
		nrerror('reallocate_iv: problem in attempt to allocate memory')
	IF (.NOT. ASSOCIATED(p)) RETURN
	nold=SIZE(p)
	reallocate_iv(1:MIN(nold,n))=p(1:MIN(nold,n))
	DEALLOCATE(p)
	END FUNCTION reallocate_iv
!BL
	FUNCTION reallocate_hv(p,n)
	CHARACTER(1), DIMENSION(:), POINTER :: p, reallocate_hv
	INTEGER(I4B), INTENT(IN) :: n
	INTEGER(I4B) :: nold,ierr
	ALLOCATE(reallocate_hv(n),stat=ierr)
	IF (ierr /= 0) CALL &
		nrerror('reallocate_hv: problem in attempt to allocate memory')
	IF (.NOT. ASSOCIATED(p)) RETURN
	nold=SIZE(p)
	reallocate_hv(1:MIN(nold,n))=p(1:MIN(nold,n))
	DEALLOCATE(p)
	END FUNCTION reallocate_hv
!BL
	FUNCTION reallocate_rm(p,n,m)
	REAL(SP), DIMENSION(:,:), POINTER :: p, reallocate_rm
	INTEGER(I4B), INTENT(IN) :: n,m
	INTEGER(I4B) :: nold,mold,ierr
	ALLOCATE(reallocate_rm(n,m),stat=ierr)
	IF (ierr /= 0) CALL &
		nrerror('reallocate_rm: problem in attempt to allocate memory')
	IF (.NOT. ASSOCIATED(p)) RETURN
	nold=SIZE(p,1)
```

```
        mold=SIZE(p,2)
        reallocate_rm(1:MIN(nold,n),1:MIN(mold,m))=&
                p(1:MIN(nold,n),1:MIN(mold,m))
        DEALLOCATE(p)
        END FUNCTION reallocate_rm
!BL
        FUNCTION reallocate_im(p,n,m)
        INTEGER(I4B), DIMENSION(:,:), POINTER :: p, reallocate_im
        INTEGER(I4B), INTENT(IN) :: n,m
        INTEGER(I4B) :: nold,mold,ierr
        ALLOCATE(reallocate_im(n,m),stat=ierr)
        IF (ierr /= 0) CALL &
                nrerror('reallocate_im: problem in attempt to allocate memory')
        IF (.NOT. ASSOCIATED(p)) RETURN
        nold=SIZE(p,1)
        mold=SIZE(p,2)
        reallocate_im(1:MIN(nold,n),1:MIN(mold,m))=&
                p(1:MIN(nold,n),1:MIN(mold,m))
        DEALLOCATE(p)
        END FUNCTION reallocate_im
!BL
        FUNCTION ifirstloc(mask)
        LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: mask
        INTEGER(I4B) :: ifirstloc
        INTEGER(I4B), DIMENSION(1) :: loc
        loc=MAXLOC(MERGE(1,0,mask))
        ifirstloc=loc(1)
        IF (.NOT. mask(ifirstloc)) ifirstloc=SIZE(mask)+1
        END FUNCTION ifirstloc
!BL
        FUNCTION imaxloc_r(arr)
        REAL(SP), DIMENSION(:), INTENT(IN) :: arr
        INTEGER(I4B) :: imaxloc_r
        INTEGER(I4B), DIMENSION(1) :: imax
        imax=MAXLOC(arr(:))
        imaxloc_r=imax(1)
        END FUNCTION imaxloc_r
!BL
        FUNCTION imaxloc_i(iarr)
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: iarr
        INTEGER(I4B), DIMENSION(1) :: imax
        INTEGER(I4B) :: imaxloc_i
        imax=MAXLOC(iarr(:))
        imaxloc_i=imax(1)
        END FUNCTION imaxloc_i
!BL
```

```
        FUNCTION iminloc(arr)
        REAL(SP), DIMENSION(:), INTENT(IN) :: arr
        INTEGER(I4B), DIMENSION(1) :: imin
        INTEGER(I4B) :: iminloc
        imin=MINLOC(arr(:))
        iminloc=imin(1)
        END FUNCTION iminloc
!BL
        SUBROUTINE assert1(n1,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        LOGICAL, INTENT(IN) :: n1
        IF (.NOT. n1) THEN
                WRITE (*,*) 'nrerror: an assertion failed with this tag:', &
                        string
                STOP 'program terminated by assert1'
        END IF
        END SUBROUTINE assert1
!BL
        SUBROUTINE assert2(n1,n2,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        LOGICAL, INTENT(IN) :: n1,n2
        IF (.NOT. (n1 .AND. n2)) THEN
                WRITE (*,*) 'nrerror: an assertion failed with this tag:', &
                        string
                STOP 'program terminated by assert2'
        END IF
        END SUBROUTINE assert2
!BL
        SUBROUTINE assert3(n1,n2,n3,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        LOGICAL, INTENT(IN) :: n1,n2,n3
        IF (.NOT. (n1 .AND. n2 .AND. n3)) THEN
                WRITE (*,*) 'nrerror: an assertion failed with this tag:', &
                        string
                STOP 'program terminated by assert3'
        END IF
        END SUBROUTINE assert3
!BL
        SUBROUTINE assert4(n1,n2,n3,n4,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        LOGICAL, INTENT(IN) :: n1,n2,n3,n4
        IF (.NOT. (n1 .AND. n2 .AND. n3 .AND. n4)) THEN
                WRITE (*,*) 'nrerror: an assertion failed with this tag:', &
                        string
                STOP 'program terminated by assert4'
        END IF
```

```
        END SUBROUTINE assert4
!BL
        SUBROUTINE assert_v(n,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        LOGICAL, DIMENSION(:), INTENT(IN) :: n
        IF (.NOT. ALL(n)) THEN
                WRITE (*,*) 'nrerror: an assertion failed with this tag:', &
                        string
                STOP 'program terminated by assert_v'
        END IF
        END SUBROUTINE assert_v
!BL
        FUNCTION assert_eq2(n1,n2,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        INTEGER, INTENT(IN) :: n1,n2
        INTEGER :: assert_eq2
        IF (n1 == n2) THEN
                assert_eq2=n1
        ELSE
                WRITE (*,*) 'nrerror: an assert_eq failed with this tag:', &
                        string
                STOP 'program terminated by assert_eq2'
        END IF
        END FUNCTION assert_eq2
!BL
        FUNCTION assert_eq3(n1,n2,n3,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        INTEGER, INTENT(IN) :: n1,n2,n3
        INTEGER :: assert_eq3
        IF (n1 == n2 .AND. n2 == n3) THEN
                assert_eq3=n1
        ELSE
                WRITE (*,*) 'nrerror: an assert_eq failed with this tag:', &
                        string
                STOP 'program terminated by assert_eq3'
        END IF
        END FUNCTION assert_eq3
!BL
        FUNCTION assert_eq4(n1,n2,n3,n4,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        INTEGER, INTENT(IN) :: n1,n2,n3,n4
        INTEGER :: assert_eq4
        IF (n1 == n2 .AND. n2 == n3 .AND. n3 == n4) THEN
                assert_eq4=n1
        ELSE
                WRITE (*,*) 'nrerror: an assert_eq failed with this tag:', &
```

```
                string
            STOP 'program terminated by assert_eq4'
        END IF
        END FUNCTION assert_eq4
!BL
        FUNCTION assert_eqn(nn,string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        INTEGER, DIMENSION(:), INTENT(IN) :: nn
        INTEGER :: assert_eqn
        IF (ALL(nn(2:) == nn(1))) THEN
            assert_eqn=nn(1)
        ELSE
            WRITE (*,*) 'nrerror: an assert_eq failed with this tag:', &
                string
            STOP 'program terminated by assert_eqn'
        END IF
        END FUNCTION assert_eqn
!BL
        SUBROUTINE nrerror(string)
        CHARACTER(LEN=*), INTENT(IN) :: string
        WRITE (*,*) 'nrerror: ',string
        STOP 'program terminated by nrerror'
        END SUBROUTINE nrerror
!BL
        FUNCTION arth_r(first,increment,n)
        REAL(SP), INTENT(IN) :: first,increment
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP), DIMENSION(n) :: arth_r
        INTEGER(I4B) :: k,k2
        REAL(SP) :: temp
        IF (n > 0) arth_r(1)=first
        IF (n <= NPAR_ARTH) THEN
            DO k=2,n
                arth_r(k)=arth_r(k-1)+increment
            END DO
        ELSE
            DO k=2,NPAR2_ARTH
                arth_r(k)=arth_r(k-1)+increment
            END DO
            temp=increment*NPAR2_ARTH
            k=NPAR2_ARTH
            DO
                IF (k >= n) EXIT
                k2=k+k
                arth_r(k+1:MIN(k2,n))=temp+arth_r(1:MIN(k,n-k))
                temp=temp+temp
```

```
            k=k2
        END DO
    END IF
    END FUNCTION arth_r
!BL
    FUNCTION arth_d(first,increment,n)
    REAL(DP), INTENT(IN) :: first,increment
    INTEGER(I4B), INTENT(IN) :: n
    REAL(DP), DIMENSION(n) :: arth_d
    INTEGER(I4B) :: k,k2
    REAL(DP) :: temp
    IF (n > 0) arth_d(1)=first
    IF (n <= NPAR_ARTH) THEN
        DO k=2,n
            arth_d(k)=arth_d(k-1)+increment
        END DO
    ELSE
        DO k=2,NPAR2_ARTH
            arth_d(k)=arth_d(k-1)+increment
        END DO
        temp=increment*NPAR2_ARTH
        k=NPAR2_ARTH
        DO
            IF (k >= n) EXIT
            k2=k+k
            arth_d(k+1:MIN(k2,n))=temp+arth_d(1:MIN(k,n-k))
            temp=temp+temp
            k=k2
        END DO
    END IF
    END FUNCTION arth_d
!BL
    FUNCTION arth_i(first,increment,n)
    INTEGER(I4B), INTENT(IN) :: first,increment,n
    INTEGER(I4B), DIMENSION(n) :: arth_i
    INTEGER(I4B) :: k,k2,temp
    IF (n > 0) arth_i(1)=first
    IF (n <= NPAR_ARTH) THEN
        DO k=2,n
            arth_i(k)=arth_i(k-1)+increment
        END DO
    ELSE
        DO k=2,NPAR2_ARTH
            arth_i(k)=arth_i(k-1)+increment
        END DO
        temp=increment*NPAR2_ARTH
```

```
            k=NPAR2_ARTH
            DO
                IF (k >= n) EXIT
                k2=k+k
                arth_i(k+1:MIN(k2,n))=temp+arth_i(1:MIN(k,n-k))
                temp=temp+temp
                k=k2
            END DO
        END IF
        END FUNCTION arth_i
!BL
!BL
        FUNCTION geop_r(first,factor,n)
        REAL(SP), INTENT(IN) :: first,factor
        INTEGER(I4B), INTENT(IN) :: n
        REAL(SP), DIMENSION(n) :: geop_r
        INTEGER(I4B) :: k,k2
        REAL(SP) :: temp
        IF (n > 0) geop_r(1)=first
        IF (n <= NPAR_GEOP) THEN
            DO k=2,n
                geop_r(k)=geop_r(k-1)*factor
            END DO
        ELSE
            DO k=2,NPAR2_GEOP
                geop_r(k)=geop_r(k-1)*factor
            END DO
            temp=factor**NPAR2_GEOP
            k=NPAR2_GEOP
            DO
                IF (k >= n) EXIT
                k2=k+k
                geop_r(k+1:MIN(k2,n))=temp*geop_r(1:MIN(k,n-k))
                temp=temp*temp
                k=k2
            END DO
        END IF
        END FUNCTION geop_r
!BL
        FUNCTION geop_d(first,factor,n)
        REAL(DP), INTENT(IN) :: first,factor
        INTEGER(I4B), INTENT(IN) :: n
        REAL(DP), DIMENSION(n) :: geop_d
        INTEGER(I4B) :: k,k2
        REAL(DP) :: temp
        IF (n > 0) geop_d(1)=first
```

```
        IF (n <= NPAR_GEOP) THEN
                DO k=2,n
                        geop_d(k)=geop_d(k-1)*factor
                END DO
        ELSE
                DO k=2,NPAR2_GEOP
                        geop_d(k)=geop_d(k-1)*factor
                END DO
                temp=factor**NPAR2_GEOP
                k=NPAR2_GEOP
                DO
                        IF (k >= n) EXIT
                        k2=k+k
                        geop_d(k+1:MIN(k2,n))=temp*geop_d(1:MIN(k,n-k))
                        temp=temp*temp
                        k=k2
                END DO
        END IF
        END FUNCTION geop_d
!BL
        FUNCTION geop_i(first,factor,n)
        INTEGER(I4B), INTENT(IN) :: first,factor,n
        INTEGER(I4B), DIMENSION(n) :: geop_i
        INTEGER(I4B) :: k,k2,temp
        IF (n > 0) geop_i(1)=first
        IF (n <= NPAR_GEOP) THEN
                DO k=2,n
                        geop_i(k)=geop_i(k-1)*factor
                END DO
        ELSE
                DO k=2,NPAR2_GEOP
                        geop_i(k)=geop_i(k-1)*factor
                END DO
                temp=factor**NPAR2_GEOP
                k=NPAR2_GEOP
                DO
                        IF (k >= n) EXIT
                        k2=k+k
                        geop_i(k+1:MIN(k2,n))=temp*geop_i(1:MIN(k,n-k))
                        temp=temp*temp
                        k=k2
                END DO
        END IF
        END FUNCTION geop_i
!BL
        FUNCTION geop_c(first,factor,n)
```

```
        COMPLEX(SP), INTENT(IN) :: first,factor
        INTEGER(I4B), INTENT(IN) :: n
        COMPLEX(SP), DIMENSION(n) :: geop_c
        INTEGER(I4B) :: k,k2
        COMPLEX(SP) :: temp
        IF (n > 0) geop_c(1)=first
        IF (n <= NPAR_GEOP) THEN
                DO k=2,n
                        geop_c(k)=geop_c(k-1)*factor
                END DO
        ELSE
                DO k=2,NPAR2_GEOP
                        geop_c(k)=geop_c(k-1)*factor
                END DO
                temp=factor**NPAR2_GEOP
                k=NPAR2_GEOP
                DO
                        IF (k >= n) EXIT
                        k2=k+k
                        geop_c(k+1:MIN(k2,n))=temp*geop_c(1:MIN(k,n-k))
                        temp=temp*temp
                        k=k2
                END DO
        END IF
        END FUNCTION geop_c
!BL
        FUNCTION geop_dv(first,factor,n)
        REAL(DP), DIMENSION(:), INTENT(IN) :: first,factor
        INTEGER(I4B), INTENT(IN) :: n
        REAL(DP), DIMENSION(SIZE(first),n) :: geop_dv
        INTEGER(I4B) :: k,k2
        REAL(DP), DIMENSION(SIZE(first)) :: temp
        IF (n > 0) geop_dv(:,1)=first(:)
        IF (n <= NPAR_GEOP) THEN
                DO k=2,n
                        geop_dv(:,k)=geop_dv(:,k-1)*factor(:)
                END DO
        ELSE
                DO k=2,NPAR2_GEOP
                        geop_dv(:,k)=geop_dv(:,k-1)*factor(:)
                END DO
                temp=factor**NPAR2_GEOP
                k=NPAR2_GEOP
                DO
                        IF (k >= n) EXIT
                        k2=k+k
```

```
                geop_dv(:,k+1:MIN(k2,n))=geop_dv(:,1:MIN(k,n-k))*&
                    SPREAD(temp,2,SIZE(geop_dv(:,1:MIN(k,n-k)),2))
                temp=temp*temp
                k=k2
            END DO
        END IF
        END FUNCTION geop_dv
!BL
!BL
        RECURSIVE FUNCTION cumsum_r(arr,seed) RESULT(ans)
        REAL(SP), DIMENSION(:), INTENT(IN) :: arr
        REAL(SP), OPTIONAL, INTENT(IN) :: seed
        REAL(SP), DIMENSION(SIZE(arr)) :: ans
        INTEGER(I4B) :: n,j
        REAL(SP) :: sd
        n=SIZE(arr)
        IF (n == 0_i4b) RETURN
        sd=0.0_sp
        IF (PRESENT(seed)) sd=seed
        ans(1)=arr(1)+sd
        IF (n < NPAR_CUMSUM) THEN
            DO j=2,n
                ans(j)=ans(j-1)+arr(j)
            END DO
        ELSE
            ans(2:n:2)=cumsum_r(arr(2:n:2)+arr(1:n-1:2),sd)
            ans(3:n:2)=ans(2:n-1:2)+arr(3:n:2)
        END IF
        END FUNCTION cumsum_r
!BL
        RECURSIVE FUNCTION cumsum_i(arr,seed) RESULT(ans)
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: arr
        INTEGER(I4B), OPTIONAL, INTENT(IN) :: seed
        INTEGER(I4B), DIMENSION(SIZE(arr)) :: ans
        INTEGER(I4B) :: n,j,sd
        n=SIZE(arr)
        IF (n == 0_i4b) RETURN
        sd=0_i4b
        IF (PRESENT(seed)) sd=seed
        ans(1)=arr(1)+sd
        IF (n < NPAR_CUMSUM) THEN
            DO j=2,n
                ans(j)=ans(j-1)+arr(j)
            END DO
        ELSE
            ans(2:n:2)=cumsum_i(arr(2:n:2)+arr(1:n-1:2),sd)
```

```
            ans(3:n:2)=ans(2:n-1:2)+arr(3:n:2)
        END IF
        END FUNCTION cumsum_i
!BL
!BL
        RECURSIVE FUNCTION cumprod(arr,seed) RESULT(ans)
        REAL(SP), DIMENSION(:), INTENT(IN) :: arr
        REAL(SP), OPTIONAL, INTENT(IN) :: seed
        REAL(SP), DIMENSION(SIZE(arr)) :: ans
        INTEGER(I4B) :: n,j
        REAL(SP) :: sd
        n=SIZE(arr)
        IF (n == 0_i4b) RETURN
        sd=1.0_sp
        IF (PRESENT(seed)) sd=seed
        ans(1)=arr(1)*sd
        IF (n < NPAR_CUMPROD) THEN
            DO j=2,n
                ans(j)=ans(j-1)*arr(j)
            END DO
        ELSE
            ans(2:n:2)=cumprod(arr(2:n:2)*arr(1:n-1:2),sd)
            ans(3:n:2)=ans(2:n-1:2)*arr(3:n:2)
        END IF
        END FUNCTION cumprod
!BL
!BL
        FUNCTION poly_rr(x,coeffs)
        REAL(SP), INTENT(IN) :: x
        REAL(SP), DIMENSION(:), INTENT(IN) :: coeffs
        REAL(SP) :: poly_rr
        REAL(SP) :: pow
        REAL(SP), DIMENSION(:), ALLOCATABLE :: vec
        INTEGER(I4B) :: i,n,nn
        n=SIZE(coeffs)
        IF (n <= 0) THEN
            poly_rr=0.0_sp
        ELSE IF (n < NPAR_POLY) THEN
            poly_rr=coeffs(n)
            DO i=n-1,1,-1
                poly_rr=x*poly_rr+coeffs(i)
            END DO
        ELSE
            ALLOCATE(vec(n+1))
            pow=x
            vec(1:n)=coeffs
```

```
            DO
                vec(n+1)=0.0_sp
                nn=ISHFT(n+1,-1)
                vec(1:nn)=vec(1:n:2)+pow*vec(2:n+1:2)
                IF (nn == 1) EXIT
                pow=pow*pow
                n=nn
            END DO
            poly_rr=vec(1)
            DEALLOCATE(vec)
        END IF
        END FUNCTION poly_rr
!BL
        FUNCTION poly_dd(x,coeffs)
        REAL(DP), INTENT(IN) :: x
        REAL(DP), DIMENSION(:), INTENT(IN) :: coeffs
        REAL(DP) :: poly_dd
        REAL(DP) :: pow
        REAL(DP), DIMENSION(:), ALLOCATABLE :: vec
        INTEGER(I4B) :: i,n,nn
        n=SIZE(coeffs)
        IF (n <= 0) THEN
            poly_dd=0.0_dp
        ELSE IF (n < NPAR_POLY) THEN
            poly_dd=coeffs(n)
            DO i=n-1,1,-1
                poly_dd=x*poly_dd+coeffs(i)
            END DO
        ELSE
            ALLOCATE(vec(n+1))
            pow=x
            vec(1:n)=coeffs
            DO
                vec(n+1)=0.0_dp
                nn=ISHFT(n+1,-1)
                vec(1:nn)=vec(1:n:2)+pow*vec(2:n+1:2)
                IF (nn == 1) EXIT
                pow=pow*pow
                n=nn
            END DO
            poly_dd=vec(1)
            DEALLOCATE(vec)
        END IF
        END FUNCTION poly_dd
!BL
        FUNCTION poly_rc(x,coeffs)
```

```
        COMPLEX(SPC), INTENT(IN) :: x
        REAL(SP), DIMENSION(:), INTENT(IN) :: coeffs
        COMPLEX(SPC) :: poly_rc
        COMPLEX(SPC) :: pow
        COMPLEX(SPC), DIMENSION(:), ALLOCATABLE :: vec
        INTEGER(I4B) :: i,n,nn
        n=SIZE(coeffs)
        IF (n <= 0) THEN
                poly_rc=0.0_sp
        ELSE IF (n < NPAR_POLY) THEN
                poly_rc=coeffs(n)
                DO i=n-1,1,-1
                        poly_rc=x*poly_rc+coeffs(i)
                END DO
        ELSE
                ALLOCATE(vec(n+1))
                pow=x
                vec(1:n)=coeffs
                DO
                        vec(n+1)=0.0_sp
                        nn=ISHFT(n+1,-1)
                        vec(1:nn)=vec(1:n:2)+pow*vec(2:n+1:2)
                        IF (nn == 1) EXIT
                        pow=pow*pow
                        n=nn
                END DO
                poly_rc=vec(1)
                DEALLOCATE(vec)
        END IF
        END FUNCTION poly_rc
!BL
        FUNCTION poly_cc(x,coeffs)
        COMPLEX(SPC), INTENT(IN) :: x
        COMPLEX(SPC), DIMENSION(:), INTENT(IN) :: coeffs
        COMPLEX(SPC) :: poly_cc
        COMPLEX(SPC) :: pow
        COMPLEX(SPC), DIMENSION(:), ALLOCATABLE :: vec
        INTEGER(I4B) :: i,n,nn
        n=SIZE(coeffs)
        IF (n <= 0) THEN
                poly_cc=0.0_sp
        ELSE IF (n < NPAR_POLY) THEN
                poly_cc=coeffs(n)
                DO i=n-1,1,-1
                        poly_cc=x*poly_cc+coeffs(i)
                END DO
```

```
        ELSE
                ALLOCATE(vec(n+1))
                pow=x
                vec(1:n)=coeffs
                DO
                        vec(n+1)=0.0_sp
                        nn=ISHFT(n+1,-1)
                        vec(1:nn)=vec(1:n:2)+pow*vec(2:n+1:2)
                        IF (nn == 1) EXIT
                        pow=pow*pow
                        n=nn
                END DO
                poly_cc=vec(1)
                DEALLOCATE(vec)
        END IF
        END FUNCTION poly_cc
!BL
        FUNCTION poly_rrv(x,coeffs)
        REAL(SP), DIMENSION(:), INTENT(IN) :: coeffs,x
        REAL(SP), DIMENSION(SIZE(x)) :: poly_rrv
        INTEGER(I4B) :: i,n,m
        m=SIZE(coeffs)
        n=SIZE(x)
        IF (m <= 0) THEN
                poly_rrv=0.0_sp
        ELSE IF (m < n .OR. m < NPAR_POLY) THEN
                poly_rrv=coeffs(m)
                DO i=m-1,1,-1
                        poly_rrv=x*poly_rrv+coeffs(i)
                END DO
        ELSE
                DO i=1,n
                        poly_rrv(i)=poly_rr(x(i),coeffs)
                END DO
        END IF
        END FUNCTION poly_rrv
!BL
        FUNCTION poly_ddv(x,coeffs)
        REAL(DP), DIMENSION(:), INTENT(IN) :: coeffs,x
        REAL(DP), DIMENSION(SIZE(x)) :: poly_ddv
        INTEGER(I4B) :: i,n,m
        m=SIZE(coeffs)
        n=SIZE(x)
        IF (m <= 0) THEN
                poly_ddv=0.0_dp
        ELSE IF (m < n .OR. m < NPAR_POLY) THEN
```

```
            poly_ddv=coeffs(m)
            DO i=m-1,1,-1
                poly_ddv=x*poly_ddv+coeffs(i)
            END DO
        ELSE
            DO i=1,n
                poly_ddv(i)=poly_dd(x(i),coeffs)
            END DO
        END IF
        END FUNCTION poly_ddv
!BL
        FUNCTION poly_msk_rrv(x,coeffs,mask)
        REAL(SP), DIMENSION(:), INTENT(IN) :: coeffs,x
        LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: mask
        REAL(SP), DIMENSION(SIZE(x)) :: poly_msk_rrv
        poly_msk_rrv=UNPACK(poly_rrv(PACK(x,mask),coeffs),mask,0.0_sp)
        END FUNCTION poly_msk_rrv
!BL
        FUNCTION poly_msk_ddv(x,coeffs,mask)
        REAL(DP), DIMENSION(:), INTENT(IN) :: coeffs,x
        LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: mask
        REAL(DP), DIMENSION(SIZE(x)) :: poly_msk_ddv
        poly_msk_ddv=UNPACK(poly_ddv(PACK(x,mask),coeffs),mask,0.0_dp)
        END FUNCTION poly_msk_ddv
!BL
!BL
        RECURSIVE FUNCTION poly_term_rr(a,b) RESULT(u)
        REAL(SP), DIMENSION(:), INTENT(IN) :: a
        REAL(SP), INTENT(IN) :: b
        REAL(SP), DIMENSION(SIZE(a)) :: u
        INTEGER(I4B) :: n,j
        n=SIZE(a)
        IF (n <= 0) RETURN
        u(1)=a(1)
        IF (n < NPAR_POLYTERM) THEN
            DO j=2,n
                u(j)=a(j)+b*u(j-1)
            END DO
        ELSE
            u(2:n:2)=poly_term_rr(a(2:n:2)+a(1:n-1:2)*b,b*b)
            u(3:n:2)=a(3:n:2)+b*u(2:n-1:2)
        END IF
        END FUNCTION poly_term_rr
!BL
        RECURSIVE FUNCTION poly_term_cc(a,b) RESULT(u)
        COMPLEX(SPC), DIMENSION(:), INTENT(IN) :: a
```

```
        COMPLEX(SPC), INTENT(IN) :: b
        COMPLEX(SPC), DIMENSION(SIZE(a)) :: u
        INTEGER(I4B) :: n,j
        n=SIZE(a)
        IF (n <= 0) RETURN
        u(1)=a(1)
        IF (n < NPAR_POLYTERM) THEN
                DO j=2,n
                        u(j)=a(j)+b*u(j-1)
                END DO
        ELSE
                u(2:n:2)=poly_term_cc(a(2:n:2)+a(1:n-1:2)*b,b*b)
                u(3:n:2)=a(3:n:2)+b*u(2:n-1:2)
        END IF
        END FUNCTION poly_term_cc
!BL
!BL
        FUNCTION zroots_unity(n,nn)
        INTEGER(I4B), INTENT(IN) :: n,nn
        COMPLEX(SPC), DIMENSION(nn) :: zroots_unity
        INTEGER(I4B) :: k
        REAL(SP) :: theta
        zroots_unity(1)=1.0
        theta=TWOPI/n
        k=1
        DO
                IF (k >= nn) EXIT
                zroots_unity(k+1)=CMPLX(COS(k*theta),SIN(k*theta),SPC)
                zroots_unity(k+2:MIN(2*k,nn))=zroots_unity(k+1)*&
                        zroots_unity(2:MIN(k,nn-k))
                k=2*k
        END DO
        END FUNCTION zroots_unity
!BL
        FUNCTION outerprod_r(a,b)
        REAL(SP), DIMENSION(:), INTENT(IN) :: a,b
        REAL(SP), DIMENSION(SIZE(a),SIZE(b)) :: outerprod_r
        outerprod_r = SPREAD(a,dim=2,ncopies=SIZE(b)) * &
                SPREAD(b,dim=1,ncopies=SIZE(a))
        END FUNCTION outerprod_r
!BL
        FUNCTION outerprod_d(a,b)
        REAL(DP), DIMENSION(:), INTENT(IN) :: a,b
        REAL(DP), DIMENSION(SIZE(a),SIZE(b)) :: outerprod_d
        outerprod_d = SPREAD(a,dim=2,ncopies=SIZE(b)) * &
                SPREAD(b,dim=1,ncopies=SIZE(a))
```

```
        END FUNCTION outerprod_d
!BL
        FUNCTION outerdiv(a,b)
        REAL(SP), DIMENSION(:), INTENT(IN) :: a,b
        REAL(SP), DIMENSION(SIZE(a),SIZE(b)) :: outerdiv
        outerdiv = SPREAD(a,dim=2,ncopies=SIZE(b)) / &
                SPREAD(b,dim=1,ncopies=SIZE(a))
        END FUNCTION outerdiv
!BL
        FUNCTION outersum(a,b)
        REAL(SP), DIMENSION(:), INTENT(IN) :: a,b
        REAL(SP), DIMENSION(SIZE(a),SIZE(b)) :: outersum
        outersum = SPREAD(a,dim=2,ncopies=SIZE(b)) + &
                SPREAD(b,dim=1,ncopies=SIZE(a))
        END FUNCTION outersum
!BL
        FUNCTION outerdiff_r(a,b)
        REAL(SP), DIMENSION(:), INTENT(IN) :: a,b
        REAL(SP), DIMENSION(SIZE(a),SIZE(b)) :: outerdiff_r
        outerdiff_r = SPREAD(a,dim=2,ncopies=SIZE(b)) - &
                SPREAD(b,dim=1,ncopies=SIZE(a))
        END FUNCTION outerdiff_r
!BL
        FUNCTION outerdiff_d(a,b)
        REAL(DP), DIMENSION(:), INTENT(IN) :: a,b
        REAL(DP), DIMENSION(SIZE(a),SIZE(b)) :: outerdiff_d
        outerdiff_d = SPREAD(a,dim=2,ncopies=SIZE(b)) - &
                SPREAD(b,dim=1,ncopies=SIZE(a))
        END FUNCTION outerdiff_d
!BL
        FUNCTION outerdiff_i(a,b)
        INTEGER(I4B), DIMENSION(:), INTENT(IN) :: a,b
        INTEGER(I4B), DIMENSION(SIZE(a),SIZE(b)) :: outerdiff_i
        outerdiff_i = SPREAD(a,dim=2,ncopies=SIZE(b)) - &
                SPREAD(b,dim=1,ncopies=SIZE(a))
        END FUNCTION outerdiff_i
!BL
        FUNCTION outerand(a,b)
        LOGICAL(LGT), DIMENSION(:), INTENT(IN) :: a,b
        LOGICAL(LGT), DIMENSION(SIZE(a),SIZE(b)) :: outerand
        outerand = SPREAD(a,dim=2,ncopies=SIZE(b)) .AND. &
                SPREAD(b,dim=1,ncopies=SIZE(a))
        END FUNCTION outerand
!BL
        SUBROUTINE scatter_add_r(dest,source,dest_index)
        REAL(SP), DIMENSION(:), INTENT(OUT) :: dest
```

```
REAL(SP), DIMENSION(:), INTENT(IN) :: source
INTEGER(I4B), DIMENSION(:), INTENT(IN) :: dest_index
INTEGER(I4B) :: m,n,j,i
n=assert_eq2(SIZE(source),SIZE(dest_index),'scatter_add_r')
m=SIZE(dest)
DO j=1,n
        i=dest_index(j)
        IF (i > 0 .AND. i <= m) dest(i)=dest(i)+source(j)
END DO
END SUBROUTINE scatter_add_r
SUBROUTINE scatter_add_d(dest,source,dest_index)
REAL(DP), DIMENSION(:), INTENT(OUT) :: dest
REAL(DP), DIMENSION(:), INTENT(IN) :: source
INTEGER(I4B), DIMENSION(:), INTENT(IN) :: dest_index
INTEGER(I4B) :: m,n,j,i
n=assert_eq2(SIZE(source),SIZE(dest_index),'scatter_add_d')
m=SIZE(dest)
DO j=1,n
        i=dest_index(j)
        IF (i > 0 .AND. i <= m) dest(i)=dest(i)+source(j)
END DO
END SUBROUTINE scatter_add_d
SUBROUTINE scatter_max_r(dest,source,dest_index)
REAL(SP), DIMENSION(:), INTENT(OUT) :: dest
REAL(SP), DIMENSION(:), INTENT(IN) :: source
INTEGER(I4B), DIMENSION(:), INTENT(IN) :: dest_index
INTEGER(I4B) :: m,n,j,i
n=assert_eq2(SIZE(source),SIZE(dest_index),'scatter_max_r')
m=SIZE(dest)
DO j=1,n
        i=dest_index(j)
        IF (i > 0 .AND. i <= m) dest(i)=MAX(dest(i),source(j))
END DO
END SUBROUTINE scatter_max_r
SUBROUTINE scatter_max_d(dest,source,dest_index)
REAL(DP), DIMENSION(:), INTENT(OUT) :: dest
REAL(DP), DIMENSION(:), INTENT(IN) :: source
INTEGER(I4B), DIMENSION(:), INTENT(IN) :: dest_index
INTEGER(I4B) :: m,n,j,i
n=assert_eq2(SIZE(source),SIZE(dest_index),'scatter_max_d')
m=SIZE(dest)
DO j=1,n
        i=dest_index(j)
        IF (i > 0 .AND. i <= m) dest(i)=MAX(dest(i),source(j))
END DO
END SUBROUTINE scatter_max_d
```

```
!BL
	SUBROUTINE diagadd_rv(mat,diag)
	REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: mat
	REAL(SP), DIMENSION(:), INTENT(IN) :: diag
	INTEGER(I4B) :: j,n
	n = assert_eq2(SIZE(diag),MIN(SIZE(mat,1),SIZE(mat,2)),'diagadd_rv')
	DO j=1,n
		mat(j,j)=mat(j,j)+diag(j)
	END DO
	END SUBROUTINE diagadd_rv
!BL
	SUBROUTINE diagadd_r(mat,diag)
	REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: mat
	REAL(SP), INTENT(IN) :: diag
	INTEGER(I4B) :: j,n
	n = MIN(SIZE(mat,1),SIZE(mat,2))
	DO j=1,n
		mat(j,j)=mat(j,j)+diag
	END DO
	END SUBROUTINE diagadd_r
!BL
	SUBROUTINE diagmult_rv(mat,diag)
	REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: mat
	REAL(SP), DIMENSION(:), INTENT(IN) :: diag
	INTEGER(I4B) :: j,n
	n = assert_eq2(SIZE(diag),MIN(SIZE(mat,1),SIZE(mat,2)),'diagmult_rv')
	DO j=1,n
		mat(j,j)=mat(j,j)*diag(j)
	END DO
	END SUBROUTINE diagmult_rv
!BL
	SUBROUTINE diagmult_r(mat,diag)
	REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: mat
	REAL(SP), INTENT(IN) :: diag
	INTEGER(I4B) :: j,n
	n = MIN(SIZE(mat,1),SIZE(mat,2))
	DO j=1,n
		mat(j,j)=mat(j,j)*diag
	END DO
	END SUBROUTINE diagmult_r
!BL
	FUNCTION get_diag_rv(mat)
	REAL(SP), DIMENSION(:,:), INTENT(IN) :: mat
	REAL(SP), DIMENSION(SIZE(mat,1)) :: get_diag_rv
	INTEGER(I4B) :: j
	j=assert_eq2(SIZE(mat,1),SIZE(mat,2),'get_diag_rv')
```

```
        DO j=1,SIZE(mat,1)
                get_diag_rv(j)=mat(j,j)
        END DO
        END FUNCTION get_diag_rv
!BL
        FUNCTION get_diag_dv(mat)
        REAL(DP), DIMENSION(:,:), INTENT(IN) :: mat
        REAL(DP), DIMENSION(SIZE(mat,1)) :: get_diag_dv
        INTEGER(I4B) :: j
        j=assert_eq2(SIZE(mat,1),SIZE(mat,2),'get_diag_dv')
        DO j=1,SIZE(mat,1)
                get_diag_dv(j)=mat(j,j)
        END DO
        END FUNCTION get_diag_dv
!BL
        SUBROUTINE put_diag_rv(diagv,mat)
        REAL(SP), DIMENSION(:), INTENT(IN) :: diagv
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: mat
        INTEGER(I4B) :: j,n
        n=assert_eq2(SIZE(diagv),MIN(SIZE(mat,1),SIZE(mat,2)),'put_diag_rv')
        DO j=1,n
                mat(j,j)=diagv(j)
        END DO
        END SUBROUTINE put_diag_rv
!BL
        SUBROUTINE put_diag_r(scal,mat)
        REAL(SP), INTENT(IN) :: scal
        REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: mat
        INTEGER(I4B) :: j,n
        n = MIN(SIZE(mat,1),SIZE(mat,2))
        DO j=1,n
                mat(j,j)=scal
        END DO
        END SUBROUTINE put_diag_r
!BL
        SUBROUTINE unit_matrix(mat)
        REAL(SP), DIMENSION(:,:), INTENT(OUT) :: mat
        INTEGER(I4B) :: i,n
        n=MIN(SIZE(mat,1),SIZE(mat,2))
        mat(:,:)=0.0_sp
        DO i=1,n
                mat(i,i)=1.0_sp
        END DO
        END SUBROUTINE unit_matrix
!BL
        FUNCTION upper_triangle(j,k,extra)
```

```
        INTEGER(I4B), INTENT(IN) :: j,k
        INTEGER(I4B), OPTIONAL, INTENT(IN) :: extra
        LOGICAL(LGT), DIMENSION(j,k) :: upper_triangle
        INTEGER(I4B) :: n
        n=0
        IF (PRESENT(extra)) n=extra
        upper_triangle=(outerdiff(arth_i(1,1,j),arth_i(1,1,k)) < n)
        END FUNCTION upper_triangle
!BL
        FUNCTION lower_triangle(j,k,extra)
        INTEGER(I4B), INTENT(IN) :: j,k
        INTEGER(I4B), OPTIONAL, INTENT(IN) :: extra
        LOGICAL(LGT), DIMENSION(j,k) :: lower_triangle
        INTEGER(I4B) :: n
        n=0
        IF (PRESENT(extra)) n=extra
        lower_triangle=(outerdiff(arth_i(1,1,j),arth_i(1,1,k)) > -n)
        END FUNCTION lower_triangle
!BL
        FUNCTION vabs(v)
        REAL(SP), DIMENSION(:), INTENT(IN) :: v
        REAL(SP) :: vabs
        vabs=SQRT(DOT_PRODUCT(v,v))
        END FUNCTION vabs
!BL
END MODULE nrutil
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/22 06:20:55 ayahil>

!!! Computation parameters held in a module for use by the minimization &
!!! other routines.

MODULE Parm

 USE Nrtype
 IMPLICIT NONE

 TYPE Submatrix_tag
   INTEGER, DIMENSION(:), POINTER :: idx
 END TYPE Submatrix_tag

 INTEGER :: iflag, k
 INTEGER, ALLOCATABLE, DIMENSION(:) :: idx
 INTEGER, POINTER :: n, p
 REAL(sp) :: norm
 REAL(sp), ALLOCATABLE, DIMENSION(:) :: diagv
 REAL(sp), POINTER, DIMENSION(:) :: w
 REAL(sp), POINTER, DIMENSION(:,:) :: r
 TYPE(Submatrix_tag), ALLOCATABLE, DIMENSION(:) :: sm

END MODULE Parm
```

```
!*****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!*****************************************
! Time-stamp: <98/06/15 20:18:51 ayahil>

!!! Generate a random premutation of the indices 1,...,n.  This routine uses
!!! completely independent random generators to avoid interfering with other
!!! random number generation.

FUNCTION Permute( n, seed ) RESULT( out )

  USE Dfport, ONLY: Time
  USE Mynr, ONLY: My_ran1
  USE My_ran_state, ONLY: My_ran_seed
  USE Nr, ONLY: Indexx
  USE Nrtype
  IMPLICIT NONE
                    ! Arguments
  INTEGER, INTENT(in) :: n
  INTEGER, OPTIONAL, INTENT(in) :: seed
  INTEGER, DIMENSION(n) :: out
                    ! Locals
  REAL(sp), DIMENSION(n) :: t
                    ! Change the seed of the random-number
                    ! generator
  IF( PRESENT(seed) ) THEN
    CALL My_ran_seed( seed )
  ELSE
    CALL My_ran_seed( Time() )
  END IF
                    ! Generate real random numbers
  CALL My_ran1( t )
                    ! The sort indices of the random numbers are
                    ! the random permutation
  CALL Indexx( t, out )

  END
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/29 11:49:28 ayahil>

!!! Prepare a matrix of mean-subtracted returns of securities in the form of
!!! percentages. Returns are provided only where prices exist for the date in
!!! question and the previous date. Otherwise, set return to -999.
!!! Securities that do not have at least CNTMN returns, making up at least a
!!! fraction PRUNE of the dates on which the securities existed are eliminated
!!! completely.

SUBROUTINE Prepare( &
   & cntmn, &          ! Minimum count of valid securities
   & idx, &            ! Index list of valid securities after pruning
   & n, &              ! # of samples
   & p, &              ! # of valid securities after pruning
   & p0, &             ! Total # of securities
   & prune, &          ! Pruning fraction
   & r, &              ! Mean-subtracted returns
   & ravg, &           ! Mean returns
   & w, &              ! Weights
   & x &               ! Prices
   & )

 USE Nrtype
 USE Nrutil, ONLY: Assert_eq, Ifirstloc
 IMPLICIT NONE
                     ! Arguments
 INTEGER, INTENT(in) :: cntmn, n, p0
 INTEGER, INTENT(out) :: p
 INTEGER, INTENT(out), DIMENSION(p0) :: idx
 REAL(sp), INTENT(in) :: prune
 REAL(sp), INTENT(in), DIMENSION(n) :: w
 REAL(sp), INTENT(in), DIMENSION(n,p0) :: x
 REAL(sp), INTENT(out), DIMENSION(n,p0) :: r
 REAL(sp), INTENT(out), DIMENSION(p0) :: ravg
                     ! Locals
 INTEGER :: chk, cnt, j
 LOGICAL, DIMENSION(n) :: msk
                     ! Check sizes
 chk = Assert_eq( n, SIZE(r,1), SIZE(w), SIZE(x,1), ' Prepare-n' )
 chk = Assert_eq( (/ p0, SIZE(idx), SIZE(r,2), SIZE(ravg), SIZE(x,2) /), &
    & ' Prepare-p0' )
```

```
                    ! Verify that the weights are non-negative
IF( ANY( w < 0.0_sp ) ) STOP 'The weights may not be negative'
                    ! Compute the returns with mean subtracted &
                    ! prune the securities with too few returns
p = 0
DO j = 1,p0
  msk = x(:,j) > 0 .AND. EOSHIFT( x(:,j), -1 ) > 0
  WHERE( msk )
    r(:,j) = 100.0_sp*(x(:,j)/EOSHIFT( x(:,j), -1 ) - 1.0_sp)
  ELSEWHERE
    r(:,j) = -999.0_sp
  END WHERE
  cnt = COUNT( msk )
  IF( cnt > MAX( REAL( cntmn, KIND=KIND(prune) ), &
     & prune*(n + 1 - Ifirstloc( x(:,j) /= 0.0_sp ) &
     & - Ifirstloc( x(n:1:-1,j) /= 0.0_sp ) ) ) ) THEN
    ravg(j) = SUM( w*r(:,j), MASK=msk )/SUM( w, MASK=msk )
    WHERE( msk )
      r(:,j) = r(:,j) - ravg(j)
    END WHERE
    p = p + 1
    idx(p) = j
  ELSE
    msk = .FALSE.
    r(:,j) = -999.0_sp
    ravg(j) = -999.0_sp
  END IF
END DO

END SUBROUTINE Prepare
```

```
                /* ****************************************
                /*  Copyright (C) 1998 by Algebron LLC.
                /*  All rights reserved.
                /*  Unauthorized reproduction prohibited.
                /* **************************************** */
                        /* Time-stamp: <98/07/24 07:59:58 ayahil> */
/*

  Prepare a matrix of mean-subtracted returns of securities in the form of
  percentages. Returns are provided only where prices exist for the date in
  question and the previous date. Otherwise, set return to -999. Securities
  that do not have at least CNTMN returns, making up at least a fraction PRUNE
  of the dates on which the securities existed, are eliminated completely.

 */

#ifndef PREPARE_H
#define PREPARE_H

void prepare_(
        int* cntmn,       /* Acceptance criterion: minimum # of returns.
                             Input [default=10] */
        int* idx,         /* Index list of accepted securities. Output
                             array(p0)*/
        int* n,           /* Total # of security samplings. Input */
        int* p,           /* # of accepted securities. Output */
        int* p0,          /* Total # of securities considered. Input */
        double* prune,    /* Acceptance criterion: minimum fraction of
                             sampled returns during the lifetime of the
                             securities. Input [default=10] */
        double* r, /* Mean subtracted returns as percentages;
                             invalid returns are set to -999. Output
                             array(n,p0) */
        double* ravg,     /* Means of valid returns, set to -999 for
                             securites which were not accepted. Output
                             array(p0) */
        double* w,        /* Time weighting of the securities in the
                             computation of the covariance matrix. Input
                             array(n) */
        double* x         /* Price history of the securities. Input
                             array (n,p0) */
        );

int cntmn_def = 10;

double prune_def = 0.2;
```

#endif

```
FUNCTION pythag_sp(a,b)
USE nrtype
IMPLICIT NONE
REAL(SP), INTENT(IN) :: a,b
REAL(SP) :: pythag_sp
REAL(SP) :: absa,absb
absa=abs(a)
absb=abs(b)
if (absa > absb) then
        pythag_sp=absa*sqrt(1.0_sp+(absb/absa)**2)
else
        if (absb == 0.0) then
                pythag_sp=0.0
        else
                pythag_sp=absb*sqrt(1.0_sp+(absa/absb)**2)
        end if
end if
END FUNCTION pythag_sp

FUNCTION pythag_dp(a,b)
USE nrtype
IMPLICIT NONE
REAL(DP), INTENT(IN) :: a,b
REAL(DP) :: pythag_dp
REAL(DP) :: absa,absb
absa=abs(a)
absb=abs(b)
if (absa > absb) then
        pythag_dp=absa*sqrt(1.0_dp+(absb/absa)**2)
else
        if (absb == 0.0) then
                pythag_dp=0.0
        else
                pythag_dp=absb*sqrt(1.0_dp+(absa/absb)**2)
        end if
end if
END FUNCTION pythag_dp
```

```
MODULE ran_state
        USE nrtype
        IMPLICIT NONE
        INTEGER, PARAMETER :: K4B=selected_int_kind(9)
        INTEGER(K4B), PARAMETER :: hg=huge(1_K4B), hgm=-hg, hgng=hgm-1
        INTEGER(K4B), SAVE :: lenran=0, seq=0
        INTEGER(K4B), SAVE :: iran0,jran0,kran0,nran0,mran0,rans
        INTEGER(K4B), DIMENSION(:,:), POINTER, SAVE :: ranseeds
        INTEGER(K4B), DIMENSION(:), POINTER, SAVE :: iran,jran,kran, &
                nran,mran,ranv
        REAL(SP), SAVE :: amm
        INTERFACE ran_hash
                MODULE PROCEDURE ran_hash_s, ran_hash_v
        END INTERFACE
CONTAINS
!BL
        SUBROUTINE ran_init(length)
        USE nrtype; USE nrutil, ONLY : arth,nrerror,reallocate
        IMPLICIT NONE
        INTEGER(K4B), INTENT(IN) :: length
        INTEGER(K4B) :: new,j,hgt
        if (length < lenran) RETURN
        hgt=hg
        if (hg /= 2147483647) call nrerror('ran_init: arith assump 1 fails')
        if (hgng >= 0)       call nrerror('ran_init: arith assump 2 fails')
        if (hgt+1 /= hgng)   call nrerror('ran_init: arith assump 3 fails')
        if (not(hg) >= 0)    call nrerror('ran_init: arith assump 4 fails')
        if (not(hgng) < 0)   call nrerror('ran_init: arith assump 5 fails')
        if (hg+hgng >= 0)    call nrerror('ran_init: arith assump 6 fails')
        if (not(-1_k4b) < 0) call nrerror('ran_init: arith assump 7 fails')
        if (not(0_k4b) >= 0) call nrerror('ran_init: arith assump 8 fails')
        if (not(1_k4b) >= 0) call nrerror('ran_init: arith assump 9 fails')
        if (lenran > 0) then
                ranseeds=>reallocate(ranseeds,length,5)
                ranv=>reallocate(ranv,length-1)
                new=lenran+1
        else
                allocate(ranseeds(length,5))
                allocate(ranv(length-1))
                new=1
                amm=nearest(1.0_sp,-1.0_sp)/hgng
                if (amm*hgng >= 1.0 .or. amm*hgng <= 0.0) &
                        call nrerror('ran_init: arth assump 10 fails')
        end if
        ranseeds(new:,1)=seq
        ranseeds(new:,2:5)=spread(arth(new,1,size(ranseeds(new:,1))),2,4)
```

```
        do j=1,4
                call ran_hash(ranseeds(new:,j),ranseeds(new:,j+1))
        end do
        where (ranseeds(new:,1:3) < 0) &
                ranseeds(new:,1:3)=not(ranseeds(new:,1:3))
        where (ranseeds(new:,4:5) == 0) ranseeds(new:,4:5)=1
        if (new == 1) then
                iran0=ranseeds(1,1)
                jran0=ranseeds(1,2)
                kran0=ranseeds(1,3)
                mran0=ranseeds(1,4)
                nran0=ranseeds(1,5)
                rans=nran0
        end if
        if (length > 1) then
                iran => ranseeds(2:,1)
                jran => ranseeds(2:,2)
                kran => ranseeds(2:,3)
                mran => ranseeds(2:,4)
                nran => ranseeds(2:,5)
                ranv = nran
        end if
        lenran=length
        END SUBROUTINE ran_init
!BL
        SUBROUTINE ran_deallocate
        if (lenran > 0) then
                deallocate(ranseeds,ranv)
                nullify(ranseeds,ranv,iran,jran,kran,mran,nran)
                lenran = 0
        end if
        END SUBROUTINE ran_deallocate
!BL
        SUBROUTINE ran_seed(sequence,size,put,get)
        IMPLICIT NONE
        INTEGER, OPTIONAL, INTENT(IN) :: sequence
        INTEGER, OPTIONAL, INTENT(OUT) :: size
        INTEGER, DIMENSION(:), OPTIONAL, INTENT(IN) :: put
        INTEGER, DIMENSION(:), OPTIONAL, INTENT(OUT) :: get
        if (present(size)) then
                size=5*lenran
        else if (present(put)) then
                if (lenran == 0) RETURN
                ranseeds=reshape(put,shape(ranseeds))
                where (ranseeds(:,1:3) < 0) ranseeds(:,1:3)=not(ranseeds(:,1:3))
                where (ranseeds(:,4:5) == 0) ranseeds(:,4:5)=1
```

```
            iran0=ranseeds(1,1)
            jran0=ranseeds(1,2)
            kran0=ranseeds(1,3)
            mran0=ranseeds(1,4)
            nran0=ranseeds(1,5)
        else if (present(get)) then
            if (lenran == 0) RETURN
            ranseeds(1,1:5)=(/ iran0,jran0,kran0,mran0,nran0 /)
            get=reshape(ranseeds,shape(get))
        else if (present(sequence)) then
            call ran_deallocate
            seq=sequence
        end if
        END SUBROUTINE ran_seed
!BL
        SUBROUTINE ran_hash_s(il,ir)
        IMPLICIT NONE
        INTEGER(K4B), INTENT(INOUT) :: il,ir
        INTEGER(K4B) :: is,j
        do j=1,4
            is=ir
            ir=ieor(ir,ishft(ir,5))+1422217823
            ir=ieor(ir,ishft(ir,-16))+1842055030
            ir=ieor(ir,ishft(ir,9))+80567781
            ir=ieor(il,ir)
            il=is
        end do
        END SUBROUTINE ran_hash_s
!BL
        SUBROUTINE ran_hash_v(il,ir)
        IMPLICIT NONE
        INTEGER(K4B), DIMENSION(:), INTENT(INOUT) :: il,ir
        INTEGER(K4B), DIMENSION(size(il)) :: is
        INTEGER(K4B) :: j
        do j=1,4
            is=ir
            ir=ieor(ir,ishft(ir,5))+1422217823
            ir=ieor(ir,ishft(ir,-16))+1842055030
            ir=ieor(ir,ishft(ir,9))+80567781
            ir=ieor(il,ir)
            il=is
        end do
        END SUBROUTINE ran_hash_v
END MODULE ran_state
```

```
SUBROUTINE ran1_s(harvest)
USE nrtype
USE ran_state, ONLY: K4B,amm,lenran,ran_init, &
        iran0,jran0,kran0,nran0,mran0,rans
IMPLICIT NONE
REAL(SP), INTENT(OUT) :: harvest
if (lenran < 1) call ran_init(1)
rans=iran0-kran0
if (rans < 0) rans=rans+2147483579_k4b
iran0=jran0
jran0=kran0
kran0=rans
nran0=ieor(nran0,ishft(nran0,13))
nran0=ieor(nran0,ishft(nran0,-17))
nran0=ieor(nran0,ishft(nran0,5))
if (nran0 == 1) nran0=270369_k4b
mran0=ieor(mran0,ishft(mran0,5))
mran0=ieor(mran0,ishft(mran0,-13))
mran0=ieor(mran0,ishft(mran0,6))
rans=ieor(nran0,rans)+mran0
harvest=amm*merge(rans,not(rans), rans<0)
END SUBROUTINE ran1_s

SUBROUTINE ran1_v(harvest)
USE nrtype
USE ran_state, ONLY: K4B,amm,lenran,ran_init, &
        iran,jran,kran,nran,mran,ranv
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(OUT) :: harvest
INTEGER(K4B) :: n
n=size(harvest)
if (lenran < n+1) call ran_init(n+1)
ranv(1:n)=iran(1:n)-kran(1:n)
where (ranv(1:n) < 0) ranv(1:n)=ranv(1:n)+2147483579_k4b
iran(1:n)=jran(1:n)
jran(1:n)=kran(1:n)
kran(1:n)=ranv(1:n)
nran(1:n)=ieor(nran(1:n),ishft(nran(1:n),13))
nran(1:n)=ieor(nran(1:n),ishft(nran(1:n),-17))
nran(1:n)=ieor(nran(1:n),ishft(nran(1:n),5))
where (nran(1:n) == 1) nran(1:n)=270369_k4b
mran(1:n)=ieor(mran(1:n),ishft(mran(1:n),5))
mran(1:n)=ieor(mran(1:n),ishft(mran(1:n),-13))
mran(1:n)=ieor(mran(1:n),ishft(mran(1:n),6))
ranv(1:n)=ieor(nran(1:n),ranv(1:n))+mran(1:n)
harvest=amm*merge(ranv(1:n),not(ranv(1:n)), ranv(1:n)<0 )
END SUBROUTINE ran1_v
```

```
                !****************************************
                ! Copyright (C) 1998 by Algebron LLC.
                ! All rights reserved.
                ! Unauthorized reproduction prohibited.
                !****************************************
                ! Time-stamp: <98/06/29 13:57:59 ayahil>

!!! Sample characteristics: standard deviation of the variables and their
!!! lambda noise estimates.

SUBROUTINE Sample( &
   & ln, &              ! Lambda noise estimates squared
   & r, &               ! Return data
   & std, &             ! Standard deviations
   & w &                ! Weights
   & )

 USE Nrtype
 USE Nrutil, ONLY: Assert_eq
 USE Parm, ONLY: n, p
 IMPLICIT NONE
                ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:) :: w
 REAL(sp), INTENT(in), DIMENSION(:,:) :: r
 REAL(sp), INTENT(out), DIMENSION(p) :: ln, std
                ! Locals
 INTEGER :: chk
 LOGICAL, DIMENSION(n,p) :: msk
                ! Check sizes
 chk = Assert_eq( n, SIZE(r,1), SIZE(w,1), ' Sample-n' )
 chk = Assert_eq( p, SIZE(ln), SIZE(std), SIZE(r,2), ' Sample-p' )
                ! Data mask
 msk = r /= -999.0_sp
                ! Verify that the weights are non-negative
 IF( ANY( w < 0.0_sp ) ) STOP 'The weights may not be negative'
                ! Sample variance of variables
 ln = SUM( SPREAD( w, DIM=2, NCOPIES=p ), DIM=1, MASK=msk )
 std = SQRT( SUM( SPREAD( w, DIM=2, NCOPIES=p )*r**2, DIM=1, MASK=msk ) &
    & /(ln + EPSILON(1.0_sp)) )
                ! Lambda noise term
 ln = ln/SQRT( (SUM( SPREAD( w**2, DIM=2, NCOPIES=p ), DIM=1, &
    & MASK=msk ) + EPSILON(1.0_sp)) )

END SUBROUTINE Sample
```

```
/*****************************************************************
* This file contains several wrapper function for the sentinel
* LM libraries.  The reason for these is that we do not have a LS_Handle
* object definition and thus can't have a fortran handle (pointer) to it.
* So, instead, a global handle to a LS_Handle object will be created
* in the object file of this program.  Every time one of the wrapper
* functions is called, that global handle is passed in.
* Another reason for these wrapper functions is the symbol table
* naming conventions used by f90.  An underscore (_) is pasted to
* function identifier names.  So we can't link directly to the
* sentinel libraries.  With these wrapper functions, we can solve the
* underscore problem.
*****************************************************************/
# include "lserv.h"
LS_HANDLE    handle;

/* Single-call licensing. */
int vlslicense_( char  *feature_name, char  *version)
{
        if (LS_SUCCESS == VLSlicense( (unsigned char *)feature_name,
        (unsigned char *) version, &handle))
                return 1;
        else
                return 0;
}
```

```
SUBROUTINE sort(arr)
USE nrtype; USE nrutil, ONLY : swap,nrerror
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(INOUT) :: arr
INTEGER(I4B), PARAMETER :: NN=15, NSTACK=50
REAL(SP) :: a
INTEGER(I4B) :: n,k,i,j,jstack,l,r
INTEGER(I4B), DIMENSION(NSTACK) :: istack
n=size(arr)
jstack=0
l=1
r=n
do
        if (r-l < NN) then
                do j=l+1,r
                        a=arr(j)
                        do i=j-1,l,-1
                                if (arr(i) <= a) exit
                                arr(i+1)=arr(i)
                        end do
                        arr(i+1)=a
                end do
                if (jstack == 0) RETURN
                r=istack(jstack)
                l=istack(jstack-1)
                jstack=jstack-2
        else
                k=(l+r)/2
                call swap(arr(k),arr(l+1))
                call swap(arr(l),arr(r),arr(l)>arr(r))
                call swap(arr(l+1),arr(r),arr(l+1)>arr(r))
                call swap(arr(l),arr(l+1),arr(l)>arr(l+1))
                i=l+1
                j=r
                a=arr(l+1)
                do
                        do
                                i=i+1
                                if (arr(i) >= a) exit
                        end do
                        do
                                j=j-1
                                if (arr(j) <= a) exit
                        end do
                        if (j < i) exit
                        call swap(arr(i),arr(j))
```

[sort.f90]

```
                end do
                arr(l+1)=arr(j)
                arr(j)=a
                jstack=jstack+2
                if (jstack > NSTACK) call nrerror('sort: NSTACK too small')
                if (r-i+1 >= j-l) then
                        istack(jstack)=r
                        istack(jstack-1)=i
                        r=j-1
                else
                        istack(jstack)=j-1
                        istack(jstack-1)=l
                        l=i
                end if
        end if
end do
END SUBROUTINE sort
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/17 21:45:37 ayahil>

!!! Optionally returns several functions of a symmetric, positive-definite,
!!! matrix.

SUBROUTINE Spd( &
  & a, &              ! Input matrix
  & adc, &            ! Cholesky decomposition of A
  & ainv, &           ! Inverse A
  & b, &              ! Input vector for solving Ax=b
  & diag, &           ! Diagonal of inverse A
  & evmin, &          ! SVD cutoff on minimum eigenvalue
  & ldet, &           ! Log-determinant of A
  & p, &              ! Diagonal of the Cholesky decomposition of A
  & x &               ! Solution of Ax=b
  & )

 USE Nr, ONLY: Choldc, Cholsl, Tqli, Tred2
 USE Nrtype
 USE Nrutil, ONLY: Assert_eq, Get_diag
 USE Utils, ONLY: Cholinv, Dmatmul_r
 IMPLICIT NONE
                    ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:,:) :: a
 REAL(sp), OPTIONAL, INTENT(in) :: evmin
 REAL(sp), OPTIONAL, INTENT(in), DIMENSION(:) :: b
 REAL(sp), OPTIONAL, INTENT(out) :: ldet
 REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(a,1)) :: diag, p, x
 REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(a,1),SIZE(a,2)) :: adc, ainv
                    ! Locals
 INTEGER :: n
 REAL(sp), DIMENSION(SIZE(a,1)) :: bb, d, e, pp
 REAL(sp), DIMENSION(SIZE(a,1),SIZE(a,2)) :: aa
                    ! Initialization
 n = Assert_eq( SIZE(a,1), SIZE(a,2), ' Spd' )
 aa = a
                    ! Singular-value decomposition (SVD)
 IF( PRESENT(evmin) ) THEN
   CALL Tred2( aa, d, e )
   CALL Tqli( d, e, aa )
   WHERE( d > evmin )
```

```
    d = 1.0_sp/d
  ELSEWHERE
    d = 0.0_sp
  END WHERE
  aa = MATMUL( Dmatmul_r( aa, d ), TRANSPOSE(aa) )
                    ! Inverse matrix
  IF( PRESENT(ainv) ) ainv = aa
                    ! Diagonal of inverse matrix
  IF( PRESENT(diag) ) diag = Get_diag( aa )
                    ! Cholesky decomposition
 ELSE
  CALL Choldc( aa, pp )
  IF( PRESENT(adc) ) adc = aa
  IF( PRESENT(p) ) p = pp
                    ! Log Determinant
  IF( PRESENT( ldet ) ) ldet = 2.0_sp*SUM( LOG( pp ) )
                    ! Full matrix inversion
  IF( PRESENT(ainv) ) THEN
    CALL Cholinv( aa, pp, ainv=ainv )
  END IF
                    ! Diagonal of inverse matrix
  IF( PRESENT(diag) ) THEN
    CALL Cholinv( aa, pp, diag=diag )
  END IF
                    ! Solution of Ax=b
  IF( PRESENT(b) .AND. PRESENT(x) ) THEN
    IF( PRESENT(ainv) ) THEN
      x = MATMUL( ainv, b )
    ELSE
      CALL Cholsl( aa, pp, b, x )
    END IF
  END IF
 END IF

END SUBROUTINE Spd
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/22 06:44:51 ayahil>

PROGRAM Test_func

  USE Dfport, ONLY: Dtime, Time
  USE Interfaces, ONLY: Covar, Func, Gen_covar, Gen_dev
  USE Nr, ONLY: Gasdev, Ran1
  USE Nrtype
  USE Nrutil, ONLY: Get_diag
  USE Parm, ONLY: diagv, iflag, k, n, norm, p, r, sm, w
  USE Ran_state, ONLY: Ran_seed
  USE Utils, ONLY: Indgen, Wher
  IMPLICIT NONE
                    ! Locals
  INTEGER :: i, igrad, j, m
  INTEGER, ALLOCATABLE, DIMENSION(:) :: idx
  LOGICAL, ALLOCATABLE, DIMENSION(:,:) :: msk
  REAL :: ta(2)
  REAL(sp) :: eps, f, f1, f2, frac, zsav
  REAL(sp), ALLOCATABLE, DIMENSION(:) :: t, xi, z
  REAL(sp), ALLOCATABLE, DIMENSION(:,:) :: c, s, v
                    ! Initialization
  ALLOCATE( n, p )
  eps = EPSILON(eps)**(1.0_sp/3.0_sp)
  CALL Ran_seed( Time() )
!!$  WRITE(*,'(a)') 'Enter frac, igrad, k, n, p: '
!!$  READ(*,*)          frac, igrad, k, n, p
  frac = 0.3
  igrad = 1
  k = 3
  n = 100
  p = 5
  ALLOCATE( c(p,p), diagv(p), idx(p), msk(n,p), r(n,p), s(p,p), &
     & sm(n), t(p), v(p,p), w(n), xi(p*k+p), z(p*k+p) )
  w = EXP( (Indgen(n) - n)/(0.5_sp*n) )
  WRITE(*,*) 'Initialization:', Dtime( ta )
                    ! Get the covariance matrix
  CALL Gen_covar( c, p )
  WRITE(*,*) 'Gen_covar:', Dtime( ta )
                    ! Get the deviates
  CALL Gen_dev( c, n, p, r )
```

```
WRITE(*,*) 'Gen_dev:', Dtime( ta )
                    ! Punch holes in the data & generate index
                    ! lists
DO i = 1,n
  CALL Ran1( t )
  msk(i,:) = ( t > frac )
  idx = Wher( msk(i,:), cnt=m )
  ALLOCATE( sm(i)%idx(m) )
  sm(i)%idx = idx(:m)
END DO
WHERE( msk )
  r = -999.0_sp
END WHERE
WRITE(*,*) 'Mask:', COUNT(msk)
                    ! Test Func
CALL Ran1( z(:p) )
CALL Gasdev( z(p+1:) )
v = Covar( z )
diagv = Get_diag( v )
WRITE(*,*) 'Initial z:', Dtime(ta)
iflag = 0
Flag_loop: DO WHILE( iflag < 2 )
  iflag = iflag + 1
  DO j = 1,100
    f = Func( z )
  END DO
  WRITE(*,*) 'Func computation:', Dtime( ta )
  DO j = 1,100
    f = Func( z, xi )
  END DO
  WRITE(*,*) 'Func with grad computation:', Dtime( ta )
                    ! Test gradient of Func
  Igrad_test: IF( igrad > 0 ) THEN
    DO j = 1,SIZE(z)
      zsav = z(j)
      z(j) = zsav + eps
      f2 = Func( z )
      z(j) = zsav - eps
      f1 = Func( z )
      z(j) = zsav
      WRITE(*,*) j, 0.5_sp/eps*(f2 - f1)/(xi(j) + TINY(1.0_sp))
    END DO
                    ! Test of diagv conservation with IFLAG=2
    Iflag_2: IF( iflag == 2 ) THEN
      WRITE(*,*) diagv
      CALL Gasdev( z(p+1:) )
```

```
        f = Func( z )
        v = Covar( z )
        diagv = Get_diag( v )
        WRITE(*,*) diagv
      END IF Iflag_2
    END IF Igrad_test
  END DO Flag_loop

END PROGRAM Test_func
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/13 13:42:54 ayahil>

PROGRAM Test_gen_covar

  USE Dfport, ONLY: Time
  USE Interfaces, ONLY: Gen_covar
  USE Nr, ONLY: Eigsrt, Tqli, Tred2
  USE Nrtype
  USE Ran_state, ONLY: Ran_seed
  IMPLICIT NONE
                    ! Locals
  INTEGER :: p
  REAL(sp), ALLOCATABLE, DIMENSION(:) :: d, e
  REAL(sp), ALLOCATABLE, DIMENSION(:,:) :: c
                    ! Initialization
  WRITE(*,'(a,$)') 'Enter p: '
  READ(*,*) p
  ALLOCATE( d(p), e(p), c(p,p) )
  CALL Ran_seed( Time() )
                    ! Get the covariance matrix
  CALL Gen_covar( c, p )
  WRITE(*,'(1p5e15.7)') TRANSPOSE(c)
  WRITE(*,*)
                    ! Compute the eigenvalues & eigenvectors
  CALL Tred2( c, d, e )
  CALL Tqli( d, e, c )
  CALL Eigsrt( d, c )
  WRITE(*,'(1p5e15.7)') d
  WRITE(*,*)
  WRITE(*,'(1p5e15.7)') TRANSPOSE(c)

END PROGRAM Test_gen_covar
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/13 13:42:47 ayahil>

PROGRAM Test_gen_dev

 USE Dfport, ONLY: Time
 USE Interfaces, ONLY: Gen_covar, Gen_dev
 USE Nrtype
 USE Ran_state, ONLY: Ran_seed
 IMPLICIT NONE
                    ! Locals
 INTEGER :: n, p
 INTEGER :: i
 REAL(sp), ALLOCATABLE, DIMENSION(:,:) :: c, x
                    ! Initialization
 WRITE(*,'(a,$)') 'Enter n, p: '
 READ(*,*) n, p
 ALLOCATE( c(p,p), x(n,p) )
 CALL Ran_seed( Time() )
                    ! Get the covariance matrix
 CALL Gen_covar( c, p )
 WRITE(*,'(1p5e15.7)') TRANSPOSE(c)
 WRITE(*,*)
                    ! Get the deviates
 CALL Gen_dev( c, n, p, x )
 c = MATMUL( TRANSPOSE(x), x )/n
 WRITE(*,'(1p5e15.7)') TRANSPOSE(c)
 WRITE(*,*)

END PROGRAM Test_gen_dev
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/26 15:58:03 ayahil>

PROGRAM Test_gof

 USE Dfport, ONLY: Time
 USE Interfaces, ONLY: Get_gof, Gen_covar, Gen_dev, Sample
 USE Nr, ONLY: Ran1, Sort, Tqli, Tred2
 USE Nrtype
 USE Parm, ONLY: n, p, sm
 USE Ran_state, ONLY: Ran_seed
 USE Utils, ONLY: Indgen, Wher
 IMPLICIT NONE
                    ! Locals
 INTEGER :: i, j, m
 INTEGER :: imc, nmc
 INTEGER, ALLOCATABLE, DIMENSION(:) :: tdx
 REAL(sp) :: gof, gof_ev, gof_std, frac, s1 ,s2, tau
 REAL(sp), ALLOCATABLE, DIMENSION(:) :: chi2, d, e, ln, prob, s, tr, w
 REAL(sp), ALLOCATABLE, DIMENSION(:,:) :: r, t, v
                    ! Initialization
 ALLOCATE( n, p )
 WRITE(*,'(a,$)') 'Enter frac, n, nmc, p, tau: '
 READ(*,*)          frac, n, nmc, p, tau
 ALLOCATE( chi2(n), d(p), e(p), ln(p), prob(n), r(n,p), s(p), sm(n), t(p,p), &
    & tdx(n), tr(n), v(p,p), w(n) )
                    ! Weights
 w = EXP( (Indgen(n) - n)/tau )
                    ! Get the covariance matrix
 CALL Gen_covar( v, p )
 WRITE(*,'(1p5e15.5)') v
 WRITE(*,*)
                    ! Compute eigenvalues
 t = v
 CALL Tred2( t, d, e, novectors=.TRUE. )
 CALL Tqli( d, e )
 CALL Sort( d )
 WRITE(*,'(1p5e15.5)') d
 WRITE(*,*)
                    ! Monte Carlo simulations with random initial
                    ! seed
 CALL Ran_seed( Time() )
```

```
Monte_Carlo: DO imc = 1,nmc
                    ! Get the deviates
  CALL Gen_dev( v, n, p, r )
                    ! Eliminate a fraction of the deviates
  DO j = 1,p
    CALL Ran1( tr )
    WHERE( tr < frac )
      r(:,j) = -999.0_sp
    END WHERE
  END DO
                    ! Index lists for valid returns by date
  DO i = 1,n
    tdx = Wher( r(i,:) /= -999.0_sp, cnt=m )
    ALLOCATE( sm(i)%idx(m) )
    sm(i)%idx = tdx(:m)
  END DO

                    ! Compute the sample characteristics
  CALL Sample( ln, r, s, w )
                    ! Compute the chi-squared
  gof = Get_gof( chi2, gof_ev, gof_std, prob, r, v, w )
  WRITE(8,*) gof
  s1 = s1 + gof
  s2 = s2 + gof**2
END DO Monte_Carlo
                    ! Mean and std of gof
s1 = s1/nmc
s2 = SQRT( (s2 - nmc*s1**2)/(nmc - 1) )
WRITE(*,*) n, gof_ev, gof_std, s1, s2

END PROGRAM Test_gof
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/17 21:45:29 ayahil>
PROGRAM Test_spd

 USE Dfport, ONLY: Time
 USE Interfaces, ONLY: Gen_covar
 USE Nr, ONLY: Ran1, Tqli, Tred2
 USE Nrtype
 USE Nrutil, ONLY: Get_diag
 USE Ran_state, ONLY: Ran_seed
 USE Utils, ONLY: Spd
 IMPLICIT NONE
                ! Parameters
 INTEGER, PARAMETER :: p=5
                ! Locals
 REAL(sp) :: ldet
 REAL(sp), DIMENSION(p) :: b, d, diag, e, x
 REAL(sp), DIMENSION(p,p) :: a, ainv, t
                ! Initialization
 CALL Ran_seed( Time() )
 CALL Gen_covar( a, p )
 CALL Ran1( b )
                ! Full matrix
 CALL Spd( a, ainv=ainv, b=b, ldet=ldet, x=x )
 WRITE(*,'(1p5e15.7)') MATMUL( ainv, a )
 WRITE(*,*)
 WRITE(*,'(1p5e15.7)') b - MATMUL( a, x )
 CALL Spd( a, b=b, x=x )
 WRITE(*,'(1p5e15.7)') b - MATMUL( a, x )
 WRITE(*,*)
 t = a
 CALL Tred2( t, d, e, novectors=.TRUE. )
 CALL Tqli( d, e )
 WRITE(*,'(1p5e15.7)') ldet, ldet - SUM( LOG( d ) ), MINVAL( d ), MAXVAL( d )
 WRITE(*,*)
                ! Eigenvalue limit
 CALL Spd( a, evmin=0.0_sp )
 WRITE(*,'(1p5e15.5)') diag, Get_diag( ainv )
 WRITE(*,*)
 WRITE(*,'(1p5e15.7)') MATMUL( ainv, a )
 WRITE(*,*)
 CALL Spd( a, ainv=ainv, diag=diag, evmin=0.5_sp )
```

```
  WRITE(*,'(1p5e15.5)') diag, Get_diag( ainv )
  WRITE(*,*)
  WRITE(*,'(1p5e15.7)') MATMUL( ainv, a )
  WRITE(*,*)

END PROGRAM Test_spd
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/06/17 21:45:26 ayahil>
PROGRAM Test_wher

 USE Utils, ONLY: Wher
 IMPLICIT NONE
                ! Parameters
 INTEGER, PARAMETER :: n=5
                ! Locals
 INTEGER :: cnt
 INTEGER, DIMENSION(n) :: i
 LOGICAL, DIMENSION(n) :: mask

 mask = (/.TRUE.,.TRUE.,.FALSE.,.FALSE.,.TRUE./)
 WRITE(*,*) mask
 WRITE(*,*) Wher(mask,cnt=cnt), cnt

END
```

```
SUBROUTINE tqli(d,e,z)
USE nrtype; USE nrutil, ONLY : assert_eq,nrerror
USE nr, ONLY : pythag
IMPLICIT NONE
REAL(SP), DIMENSION(:), INTENT(INOUT) :: d,e
REAL(SP), DIMENSION(:,:), OPTIONAL, INTENT(INOUT) :: z
INTEGER(I4B) :: i,iter,l,m,n,ndum
REAL(SP) :: b,c,dd,f,g,p,r,s
REAL(SP), DIMENSION(size(e)) :: ff
n=assert_eq(size(d),size(e),'tqli: n')
if (present(z)) ndum=assert_eq(n,size(z,1),size(z,2),'tqli: ndum')
e(:)=eoshift(e(:),1)
do l=1,n
        iter=0
        iterate: do
                do m=l,n-1
                        dd=abs(d(m))+abs(d(m+1))
                        if (abs(e(m))+dd == dd) exit
                end do
                if (m == l) exit iterate
                if (iter == 30) call nrerror('too many iterations in tqli')
                iter=iter+1
                g=(d(l+1)-d(l))/(2.0_sp*e(l))
                r=pythag(g,1.0_sp)
                g=d(m)-d(l)+e(l)/(g+sign(r,g))
                s=1.0
                c=1.0
                p=0.0
                do i=m-1,l,-1
                        f=s*e(i)
                        b=c*e(i)
                        r=pythag(f,g)
                        e(i+1)=r
                        if (r == 0.0) then
                                d(i+1)=d(i+1)-p
                                e(m)=0.0
                                cycle iterate
                        end if
                        s=f/r
                        c=g/r
                        g=d(i+1)-p
                        r=(d(i)-g)*s+2.0_sp*c*b
                        p=s*r
                        d(i+1)=g+p
                        g=c*r-b
                        if (present(z)) then
```

```
                    ff(1:n)=z(1:n,i+1)
                    z(1:n,i+1)=s*z(1:n,i)+c*ff(1:n)
                    z(1:n,i)=c*z(1:n,i)-s*ff(1:n)
                end if
            end do
            d(l)=d(l)-p
            e(l)=g
            e(m)=0.0
        end do iterate
    end do
    END SUBROUTINE tqli
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/05/06 10:54:12 ayahil>

FUNCTION Trace( a ) RESULT( out )

 USE Nrtype
 USE Nrutil, ONLY: Get_diag
 IMPLICIT NONE
                 ! Arguments
 REAL(sp), INTENT(in), DIMENSION(:,:) :: a
 REAL(sp) :: out
                 ! Locals
 out = SUM( Get_diag( a ) )

END FUNCTION Trace
```

```
SUBROUTINE tred2(a,d,e,novectors)
USE nrtype; USE nrutil, ONLY : assert_eq,outerprod
IMPLICIT NONE
REAL(SP), DIMENSION(:,:), INTENT(INOUT) :: a
REAL(SP), DIMENSION(:), INTENT(OUT) :: d,e
LOGICAL(LGT), OPTIONAL, INTENT(IN) :: novectors
INTEGER(I4B) :: i,j,l,n
REAL(SP) :: f,g,h,hh,scale
REAL(SP), DIMENSION(size(a,1)) :: gg
LOGICAL(LGT), SAVE :: yesvec=.true.
n=assert_eq(size(a,1),size(a,2),size(d),size(e),'tred2')
if (present(novectors)) then
  yesvec=.not. novectors
else
  yesvec=.true.
end if
  do i=n,2,-1
        l=i-1
        h=0.0
        if (l > 1) then
              scale=sum(abs(a(i,1:l)))
              if (scale == 0.0) then
                    e(i)=a(i,l)
              else
                    a(i,1:l)=a(i,1:l)/scale
                    h=sum(a(i,1:l)**2)
                    f=a(i,l)
                    g=-sign(sqrt(h),f)
                    e(i)=scale*g
                    h=h-f*g
                    a(i,l)=f-g
                    if (yesvec) a(1:l,i)=a(i,1:l)/h
                    do j=1,l
                          e(j)=(dot_product(a(j,1:j),a(i,1:j)) &
                          +dot_product(a(j+1:l,j),a(i,j+1:l)))/h
                    end do
                    f=dot_product(e(1:l),a(i,1:l))
                    hh=f/(h+h)
                    e(1:l)=e(1:l)-hh*a(i,1:l)
                    do j=1,l
                          a(j,1:j)=a(j,1:j)-a(i,j)*e(1:j)-e(j)*a(i,1:j)
                    end do
              end if
        else
              e(i)=a(i,l)
        end if
```

```
        d(i)=h
    end do
    if (yesvec) d(1)=0.0
    e(1)=0.0
    do i=1,n
        if (yesvec) then
            l=i-1
            if (d(i) /= 0.0) then
                gg(1:l)=matmul(a(i,1:l),a(1:l,1:l))
                a(1:l,1:l)=a(1:l,1:l)-outerprod(a(1:l,i),gg(1:l))
            end if
            d(i)=a(i,i)
            a(i,i)=1.0
            a(i,1:l)=0.0
            a(1:l,i)=0.0
        else
            d(i)=a(i,i)
        end if
    end do
    END SUBROUTINE tred2
```

```
!****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!****************************************
! Time-stamp: <98/07/08 09:34:32 ayahil>

!!! Interfaces for utility routines.

MODULE Utils

 INTERFACE
   SUBROUTINE Bootstrap( freq, n, seed )
    INTEGER, INTENT(in) :: n, seed
    INTEGER, INTENT(out), DIMENSION(n) :: freq
   END SUBROUTINE Bootstrap
 END INTERFACE

 INTERFACE
   SUBROUTINE Cholinv( a, p, ainv, diag )
    USE Nrtype
    REAL(sp), INTENT(in), DIMENSION(:,:) :: a
    REAL(sp), INTENT(in), DIMENSION(:) :: p
    REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(a,1),SIZE(a,2)) :: ainv
    REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(a,1)) :: diag
   END SUBROUTINE Cholinv
 END INTERFACE

 INTERFACE
   FUNCTION Dmatmul_l( diag, mat ) RESULT( out )
    USE Nrtype
    REAL(sp), INTENT(in), DIMENSION(:) :: diag
    REAL(sp), INTENT(in), DIMENSION(:,:) :: mat
    REAL(sp), DIMENSION(SIZE(diag),SIZE(mat,2)) :: out
   END FUNCTION Dmatmul_l
 END INTERFACE

 INTERFACE
   FUNCTION Dmatmul_r( mat, diag ) RESULT( out )
    USE Nrtype
    REAL(sp), INTENT(in), DIMENSION(:) :: diag
    REAL(sp), INTENT(in), DIMENSION(:,:) :: mat
    REAL(sp), DIMENSION(SIZE(mat,1),SIZE(diag)) :: out
   END FUNCTION Dmatmul_r
 END INTERFACE
```

```
INTERFACE
  FUNCTION Findgen( n ) RESULT( out )
    USE Nrtype
    INTEGER, INTENT(in) :: n
    REAL(sp), DIMENSION(n) :: out
  END FUNCTION Findgen
END INTERFACE

INTERFACE
  FUNCTION Indgen( n ) RESULT( out )
    INTEGER, INTENT(in) :: n
    INTEGER, DIMENSION(n) :: out
  END FUNCTION Indgen
END INTERFACE

INTERFACE
  FUNCTION Permute( n, seed ) RESULT( out )
    USE Nrtype
    INTEGER, INTENT(in) :: n
    INTEGER, OPTIONAL :: seed
    INTEGER, DIMENSION(n) :: out
  END FUNCTION Permute
END INTERFACE

INTERFACE
  SUBROUTINE Spd( a, adc, ainv, b, diag, evmin, ldet, p, x )
    USE Nrtype
    REAL(sp), INTENT(in), DIMENSION(:,:) :: a
    REAL(sp), OPTIONAL, INTENT(in) :: evmin
    REAL(sp), OPTIONAL, INTENT(in), DIMENSION(:) :: b
    REAL(sp), OPTIONAL, INTENT(out) :: ldet
    REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(a,1)) :: diag, p, x
    REAL(sp), OPTIONAL, INTENT(out), DIMENSION(SIZE(a,1),SIZE(a,2)) :: &
      adc, ainv
  END SUBROUTINE Spd
END INTERFACE

INTERFACE
  FUNCTION Trace( a ) RESULT( out )
    USE Nrtype
    REAL(sp), INTENT(in), DIMENSION(:,:) :: a
    REAL(sp) :: out
  END FUNCTION Trace
END INTERFACE

INTERFACE Wher
```

```
    FUNCTION Wher_1( mask, cnt ) RESULT( out )
      INTEGER, OPTIONAL, INTENT(out) :: cnt
      LOGICAL, INTENT(in), DIMENSION(:) :: mask
      INTEGER, DIMENSION(SIZE(mask)) :: out
    END FUNCTION Wher_1

    FUNCTION Wher_2( mask, cnt ) RESULT( out )
      INTEGER, OPTIONAL, INTENT(out) :: cnt
      LOGICAL, INTENT(in), DIMENSION(:,:) :: mask
      INTEGER, DIMENSION(SIZE(mask)) :: out
    END FUNCTION Wher_2

    FUNCTION Wher_3( mask, cnt ) RESULT( out )
      INTEGER, OPTIONAL, INTENT(out) :: cnt
      LOGICAL, INTENT(in), DIMENSION(:,:,:) :: mask
      INTEGER, DIMENSION(SIZE(mask)) :: out
    END FUNCTION Wher_3
  END INTERFACE

END MODULE Utils
```

```
!*****************************************
! Copyright (C) 1998 by Algebron LLC.
! All rights reserved.
! Unauthorized reproduction prohibited.
!*****************************************
! Time-stamp: <98/06/17 21:46:21 ayahil>

!!! IDL-like WHERE function, returning the 1-d indices where MASK is true.
!!! Note that, unlike in IDL, the size of the output array is always equal to
!!! the size of MASK, with the trailing indices filled with SIZE(MASK)+1.

FUNCTION Wher_1( mask, cnt ) RESULT( out )

 USE Utils, ONLY: Indgen
 IMPLICIT NONE
                 ! Arguments
 INTEGER, INTENT(out), OPTIONAL :: cnt
 LOGICAL, INTENT(in), DIMENSION(:) :: mask
 INTEGER, DIMENSION(SIZE(mask)) :: out
                 ! Locals
 INTEGER :: i

 out = PACK( RESHAPE(Indgen(SIZE(mask)),SHAPE(mask)), MASK=mask, &
    & VECTOR=SPREAD(SIZE(mask)+1,DIM=1,NCOPIES=SIZE(mask)) )
 IF( PRESENT(cnt) ) THEN
   cnt = COUNT(mask)
 END IF

END FUNCTION Wher_1

FUNCTION Wher_2( mask, cnt ) RESULT( out )

 USE Utils, ONLY: Indgen
 IMPLICIT NONE
                 ! Arguments
 INTEGER, INTENT(out), OPTIONAL :: cnt
 LOGICAL, INTENT(in), DIMENSION(:,:) :: mask
 INTEGER, DIMENSION(SIZE(mask)) :: out
                 ! Locals
 INTEGER :: i

 out = PACK( RESHAPE(Indgen(SIZE(mask)),SHAPE(mask)), MASK=mask, &
    & VECTOR=SPREAD(SIZE(mask)+1,DIM=1,NCOPIES=SIZE(mask)) )
 IF( PRESENT(cnt) ) THEN
   cnt = COUNT(mask)
 END IF
```

```
END FUNCTION Wher_2

FUNCTION Wher_3( mask, cnt ) RESULT( out )

 USE Utils, ONLY: Indgen
 IMPLICIT NONE
                    ! Arguments
 INTEGER, INTENT(out), OPTIONAL :: cnt
 LOGICAL, INTENT(in), DIMENSION(:,:,:) :: mask
 INTEGER, DIMENSION(SIZE(mask)) :: out
                    ! Locals
 INTEGER :: i

 out = PACK( RESHAPE(Indgen(SIZE(mask)),SHAPE(mask)), MASK=mask, &
    & VECTOR=SPREAD(SIZE(mask)+1,DIM=1,NCOPIES=SIZE(mask)) )
 IF( PRESENT(cnt) ) THEN
   cnt = COUNT(mask)
 END IF

END FUNCTION Wher_3
```